

Document Number: MCUXSDKAPIRM
Rev 2.12.0
Jul 2022

MCUXpresso SDK API Reference Manual

NXP Semiconductors



Contents

Chapter 1 Introduction

Chapter 2 Trademarks

Chapter 3 Architectural Overview

Chapter 4 Clock Driver

4.1 Overview	7
4.2 Data Structure Documentation	15
4.2.1 struct sim_clock_config_t	15
4.2.2 struct oscer_config_t	15
4.2.3 struct osc_config_t	16
4.2.4 struct mcg_pll_config_t	16
4.2.5 struct mcg_config_t	17
4.3 Macro Definition Documentation	18
4.3.1 MCG_CONFIG_CHECK_PARAM	18
4.3.2 FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL	18
4.3.3 FSL_CLOCK_DRIVER_VERSION	18
4.3.4 MCG_INTERNAL_IRC_48M	18
4.3.5 DMAMUX_CLOCKS	18
4.3.6 RTC_CLOCKS	19
4.3.7 SAI_CLOCKS	19
4.3.8 PORT_CLOCKS	19
4.3.9 FLEXBUS_CLOCKS	19
4.3.10 EWM_CLOCKS	19
4.3.11 PIT_CLOCKS	20
4.3.12 DSPI_CLOCKS	20
4.3.13 LPTMR_CLOCKS	20
4.3.14 FTM_CLOCKS	20
4.3.15 EDMA_CLOCKS	20
4.3.16 LPUART_CLOCKS	21
4.3.17 DAC_CLOCKS	21
4.3.18 ADC16_CLOCKS	21
4.3.19 VREF_CLOCKS	21
4.3.20 UART_CLOCKS	21

Section No.	Title	Page No.
4.3.21	RNGA_CLOCKS	22
4.3.22	CRC_CLOCKS	22
4.3.23	I2C_CLOCKS	22
4.3.24	FTF_CLOCKS	22
4.3.25	PDB_CLOCKS	22
4.3.26	CMP_CLOCKS	23
4.3.27	SYS_CLK	23
4.4	Enumeration Type Documentation	23
4.4.1	clock_name_t	23
4.4.2	clock_usb_src_t	23
4.4.3	clock_ip_name_t	24
4.4.4	osc_mode_t	24
4.4.5	_osc_cap_load	24
4.4.6	_oscer_enable_mode	24
4.4.7	mcg_fll_src_t	24
4.4.8	mcg_irc_mode_t	24
4.4.9	mcg_dmx32_t	25
4.4.10	mcg_drs_t	25
4.4.11	mcg_pll_ref_src_t	25
4.4.12	mcg_clkout_src_t	25
4.4.13	mcg_atm_select_t	25
4.4.14	mcg_oscsel_t	26
4.4.15	mcg_pll_clk_select_t	26
4.4.16	mcg_monitor_mode_t	26
4.4.17	anonymous enum	26
4.4.18	anonymous enum	26
4.4.19	anonymous enum	27
4.4.20	anonymous enum	27
4.4.21	mcg_mode_t	27
4.5	Function Documentation	27
4.5.1	CLOCK_EnableClock	27
4.5.2	CLOCK_DisableClock	28
4.5.3	CLOCK_SetLuartClock	28
4.5.4	CLOCK_SetEr32kClock	28
4.5.5	CLOCK_SetTraceClock	28
4.5.6	CLOCK_SetPllFllSelClock	28
4.5.7	CLOCK_SetClkOutClock	29
4.5.8	CLOCK_SetRtcClkOutClock	29
4.5.9	CLOCK_EnableUsbfs0Clock	29
4.5.10	CLOCK_DisableUsbfs0Clock	29
4.5.11	CLOCK_SetOutDiv	30
4.5.12	CLOCK_GetFreq	31
4.5.13	CLOCK_GetCoreSysClkFreq	31

Section No.	Title	Page No.
4.5.14	CLOCK_GetPlatClkFreq	31
4.5.15	CLOCK_GetBusClkFreq	31
4.5.16	CLOCK_GetFlexBusClkFreq	32
4.5.17	CLOCK_GetFlashClkFreq	32
4.5.18	CLOCK_GetPllFllSelClkFreq	32
4.5.19	CLOCK_GetEr32kClkFreq	32
4.5.20	CLOCK_GetOsc0ErClkUndivFreq	32
4.5.21	CLOCK_GetOsc0ErClkFreq	32
4.5.22	CLOCK_GetOsc0ErClkDivFreq	33
4.5.23	CLOCK_SetSimConfig	33
4.5.24	CLOCK_SetSimSafeDivs	33
4.5.25	CLOCK_GetOutClkFreq	33
4.5.26	CLOCK_GetFllFreq	33
4.5.27	CLOCK_GetInternalRefClkFreq	34
4.5.28	CLOCK_GetFixedFreqClkFreq	34
4.5.29	CLOCK_GetPll0Freq	34
4.5.30	CLOCK_SetLowPowerEnable	34
4.5.31	CLOCK_SetInternalRefClkConfig	35
4.5.32	CLOCK_SetExternalRefClkConfig	35
4.5.33	CLOCK_SetFllExtRefDiv	36
4.5.34	CLOCK_EnablePll0	36
4.5.35	CLOCK_DisablePll0	36
4.5.36	CLOCK_CalcPllDiv	36
4.5.37	CLOCK_SetOsc0MonitorMode	37
4.5.38	CLOCK_SetRtcOscMonitorMode	37
4.5.39	CLOCK_SetPll0MonitorMode	37
4.5.40	CLOCK_GetStatusFlags	37
4.5.41	CLOCK_ClearStatusFlags	38
4.5.42	OSC_SetExtRefClkConfig	38
4.5.43	OSC_SetCapLoad	39
4.5.44	CLOCK_InitOsc0	39
4.5.45	CLOCK_DeinitOsc0	39
4.5.46	CLOCK_SetXtal0Freq	39
4.5.47	CLOCK_SetXtal32Freq	40
4.5.48	CLOCK_SetSlowIrcFreq	40
4.5.49	CLOCK_SetFastIrcFreq	40
4.5.50	CLOCK_TrimInternalRefClk	40
4.5.51	CLOCK_GetMode	41
4.5.52	CLOCK_SetFeiMode	41
4.5.53	CLOCK_SetFeeMode	42
4.5.54	CLOCK_SetFbiMode	42
4.5.55	CLOCK_SetFbeMode	43
4.5.56	CLOCK_SetBlpiMode	44
4.5.57	CLOCK_SetBlpeMode	44
4.5.58	CLOCK_SetPbeMode	44

Section No.	Title	Page No.
4.5.59	CLOCK_SetPeeMode	45
4.5.60	CLOCK_ExternalModeToFbeModeQuick	45
4.5.61	CLOCK_InternalModeToFbiModeQuick	46
4.5.62	CLOCK_BootToFeiMode	46
4.5.63	CLOCK_BootToFeeMode	47
4.5.64	CLOCK_BootToBlpiMode	47
4.5.65	CLOCK_BootToBlpeMode	48
4.5.66	CLOCK_BootToPeeMode	48
4.5.67	CLOCK_SetMcgConfig	49
4.6	Variable Documentation	49
4.6.1	g_xtal0Freq	49
4.6.2	g_xtal32Freq	50
4.7	Multipurpose Clock Generator (MCG)	51
4.7.1	Function description	51
4.7.2	Typical use case	53
4.7.3	Code Configuration Option	56

Chapter 5 ADC16: 16-bit SAR Analog-to-Digital Converter Driver

5.1	Overview	57
5.2	Typical use case	57
5.2.1	Polling Configuration	57
5.2.2	Interrupt Configuration	57
5.3	Data Structure Documentation	60
5.3.1	struct adc16_config_t	60
5.3.2	struct adc16_hardware_compare_config_t	61
5.3.3	struct adc16_channel_config_t	61
5.4	Macro Definition Documentation	61
5.4.1	FSL_ADC16_DRIVER_VERSION	62
5.5	Enumeration Type Documentation	62
5.5.1	_adc16_channel_status_flags	62
5.5.2	_adc16_status_flags	62
5.5.3	adc16_channel_mux_mode_t	62
5.5.4	adc16_clock_divider_t	62
5.5.5	adc16_resolution_t	62
5.5.6	adc16_clock_source_t	63
5.5.7	adc16_long_sample_mode_t	63
5.5.8	adc16_reference_voltage_source_t	63
5.5.9	adc16_hardware_average_mode_t	64
5.5.10	adc16_hardware_compare_mode_t	64

Section No.	Title	Page No.
5.6 Function Documentation		64
5.6.1 ADC16_Init		64
5.6.2 ADC16_Deinit		64
5.6.3 ADC16_GetDefaultConfig		64
5.6.4 ADC16_DoAutoCalibration		65
5.6.5 ADC16_SetOffsetValue		65
5.6.6 ADC16_EnableDMA		66
5.6.7 ADC16_EnableHardwareTrigger		66
5.6.8 ADC16_SetChannelMuxMode		66
5.6.9 ADC16_SetHardwareCompareConfig		67
5.6.10 ADC16_SetHardwareAverage		67
5.6.11 ADC16_GetStatusFlags		67
5.6.12 ADC16_ClearStatusFlags		67
5.6.13 ADC16_EnableAsynchronousClockOutput		68
5.6.14 ADC16_SetChannelConfig		68
5.6.15 ADC16_GetChannelConversionValue		69
5.6.16 ADC16_GetChannelStatusFlags		69

Chapter 6 CMP: Analog Comparator Driver

6.1 Overview		70
6.2 Typical use case		70
6.2.1 Polling Configuration		70
6.2.2 Interrupt Configuration		70
6.3 Data Structure Documentation		72
6.3.1 struct cmp_config_t		72
6.3.2 struct cmp_filter_config_t		72
6.3.3 struct cmp_dac_config_t		73
6.4 Macro Definition Documentation		73
6.4.1 FSL_CMP_DRIVER_VERSION		73
6.5 Enumeration Type Documentation		73
6.5.1 _cmp_interrupt_enable		73
6.5.2 _cmp_status_flags		73
6.5.3 cmp_hysteresis_mode_t		74
6.5.4 cmp_reference_voltage_source_t		74
6.6 Function Documentation		74
6.6.1 CMP_Init		74
6.6.2 CMP_Deinit		74
6.6.3 CMP_Enable		76
6.6.4 CMP_GetDefaultConfig		76

Section No.	Title	Page No.
6.6.5	CMP_SetInputChannels	76
6.6.6	CMP_EnableDMA	77
6.6.7	CMP_EnableWindowMode	77
6.6.8	CMP_SetFilterConfig	77
6.6.9	CMP_SetDACConfig	77
6.6.10	CMP_EnableInterrupts	78
6.6.11	CMP_DisableInterrupts	78
6.6.12	CMP_GetStatusFlags	78
6.6.13	CMP_ClearStatusFlags	78

Chapter 7 Common Driver

7.1	Overview	80
7.2	Macro Definition Documentation	82
7.2.1	FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ	82
7.2.2	MAKE_STATUS	82
7.2.3	MAKE_VERSION	83
7.2.4	FSL_COMMON_DRIVER_VERSION	83
7.2.5	DEBUG_CONSOLE_DEVICE_TYPE_NONE	83
7.2.6	DEBUG_CONSOLE_DEVICE_TYPE_UART	83
7.2.7	DEBUG_CONSOLE_DEVICE_TYPE_LPUART	83
7.2.8	DEBUG_CONSOLE_DEVICE_TYPE_LPSCI	83
7.2.9	DEBUG_CONSOLE_DEVICE_TYPE_USBCDC	83
7.2.10	DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM	83
7.2.11	DEBUG_CONSOLE_DEVICE_TYPE_IUART	83
7.2.12	DEBUG_CONSOLE_DEVICE_TYPE_VUSART	83
7.2.13	DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART	83
7.2.14	DEBUG_CONSOLE_DEVICE_TYPE_SWO	83
7.2.15	DEBUG_CONSOLE_DEVICE_TYPE_QSCI	83
7.2.16	ARRAY_SIZE	83
7.3	Typedef Documentation	83
7.3.1	status_t	83
7.4	Enumeration Type Documentation	84
7.4.1	_status_groups	84
7.4.2	anonymous enum	86
7.5	Function Documentation	87
7.5.1	SDK_Malloc	87
7.5.2	SDK_Free	87
7.5.3	SDK_DelayAtLeastUs	87

Section No.	Title	Page No.
Chapter 8 CRC: Cyclic Redundancy Check Driver		
8.1	Overview	88
8.2	CRC Driver Initialization and Configuration	88
8.3	CRC Write Data	88
8.4	CRC Get Checksum	88
8.5	Comments about API usage in RTOS	89
8.6	Data Structure Documentation	90
8.6.1	struct crc_config_t	90
8.7	Macro Definition Documentation	91
8.7.1	FSL_CRC_DRIVER_VERSION	91
8.7.2	CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT	91
8.8	Enumeration Type Documentation	91
8.8.1	crc_bits_t	91
8.8.2	crc_result_t	91
8.9	Function Documentation	91
8.9.1	CRC_Init	91
8.9.2	CRC_Deinit	92
8.9.3	CRC_GetDefaultConfig	92
8.9.4	CRC_WriteData	92
8.9.5	CRC_Get32bitResult	93
8.9.6	CRC_Get16bitResult	93
Chapter 9 DAC: Digital-to-Analog Converter Driver		
9.1	Overview	94
9.2	Typical use case	94
9.2.1	Working as a basic DAC without the hardware buffer feature	94
9.2.2	Working with the hardware buffer	94
9.3	Data Structure Documentation	96
9.3.1	struct dac_config_t	96
9.3.2	struct dac_buffer_config_t	96
9.4	Macro Definition Documentation	97
9.4.1	FSL_DAC_DRIVER_VERSION	97
9.5	Enumeration Type Documentation	97

Section No.	Title	Page No.
9.5.1	_dac_buffer_status_flags	97
9.5.2	_dac_buffer_interrupt_enable	97
9.5.3	dac_reference_voltage_source_t	97
9.5.4	dac_buffer_trigger_mode_t	98
9.5.5	dac_buffer_watermark_t	98
9.5.6	dac_buffer_work_mode_t	98
9.6	Function Documentation	98
9.6.1	DAC_Init	98
9.6.2	DAC_Deinit	99
9.6.3	DAC_GetDefaultConfig	99
9.6.4	DAC_Enable	99
9.6.5	DAC_EnableBuffer	99
9.6.6	DAC_SetBufferConfig	100
9.6.7	DAC_GetDefaultBufferConfig	100
9.6.8	DAC_EnableBufferDMA	100
9.6.9	DAC_SetBufferValue	101
9.6.10	DAC_DoSoftwareTriggerBuffer	101
9.6.11	DAC_GetBufferReadPointer	101
9.6.12	DAC_SetBufferReadPointer	102
9.6.13	DAC_EnableBufferInterrupts	102
9.6.14	DAC_DisableBufferInterrupts	102
9.6.15	DAC_GetBufferStatusFlags	102
9.6.16	DAC_ClearBufferStatusFlags	103

Chapter 10 DMAMUX: Direct Memory Access Multiplexer Driver

10.1	Overview	104
10.2	Typical use case	104
10.2.1	DMAMUX Operation	104
10.3	Macro Definition Documentation	104
10.3.1	FSL_DMAMUX_DRIVER_VERSION	104
10.4	Function Documentation	104
10.4.1	DMAMUX_Init	105
10.4.2	DMAMUX_Deinit	106
10.4.3	DMAMUX_EnableChannel	106
10.4.4	DMAMUX_DisableChannel	106
10.4.5	DMAMUX_SetSource	107
10.4.6	DMAMUX_EnablePeriodTrigger	107
10.4.7	DMAMUX_DisablePeriodTrigger	107

Section No.	Title	Page No.
Chapter 11 DSPI: Serial Peripheral Interface Driver		
11.1	Overview	108
11.2	DSPI Driver	109
11.2.1	Overview	109
11.2.2	Typical use case	109
11.2.3	Data Structure Documentation	116
11.2.4	Macro Definition Documentation	123
11.2.5	Typedef Documentation	123
11.2.6	Enumeration Type Documentation	124
11.2.7	Function Documentation	128
11.2.8	Variable Documentation	147
11.3	DSPI eDMA Driver	148
11.3.1	Overview	148
11.3.2	Data Structure Documentation	149
11.3.3	Macro Definition Documentation	152
11.3.4	Typedef Documentation	152
11.3.5	Function Documentation	153
11.4	DSPI FreeRTOS Driver	159
11.4.1	Overview	159
11.4.2	Macro Definition Documentation	159
11.4.3	Function Documentation	159
11.5	DSPI CMSIS Driver	162
11.5.1	Function groups	162
11.5.2	Typical use case	163
Chapter 12 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver		
12.1	Overview	164
12.2	Typical use case	164
12.2.1	eDMA Operation	164
12.3	Data Structure Documentation	169
12.3.1	struct edma_config_t	169
12.3.2	struct edma_transfer_config_t	170
12.3.3	struct edma_channel_Preemption_config_t	171
12.3.4	struct edma_minor_offset_config_t	171
12.3.5	struct edma_tcd_t	171
12.3.6	struct edma_handle_t	172
12.4	Macro Definition Documentation	173

Section No.	Title	Page No.
12.4.1	FSL_EDMA_DRIVER_VERSION	173
12.5	Typedef Documentation	173
12.5.1	edma_callback	173
12.6	Enumeration Type Documentation	174
12.6.1	edma_transfer_size_t	174
12.6.2	edma_modulo_t	174
12.6.3	edma_bandwidth_t	175
12.6.4	edma_channel_link_type_t	175
12.6.5	anonymous enum	175
12.6.6	anonymous enum	176
12.6.7	edma_interrupt_enable_t	176
12.6.8	edma_transfer_type_t	176
12.6.9	anonymous enum	176
12.7	Function Documentation	177
12.7.1	EDMA_Init	177
12.7.2	EDMA_Deinit	178
12.7.3	EDMA_InstallTCD	178
12.7.4	EDMA_GetDefaultConfig	178
12.7.5	EDMA_EnableContinuousChannelLinkMode	179
12.7.6	EDMA_EnableMinorLoopMapping	179
12.7.7	EDMA_ResetChannel	179
12.7.8	EDMA_SetTransferConfig	180
12.7.9	EDMA_SetMinorOffsetConfig	180
12.7.10	EDMA_SetChannelPreemptionConfig	181
12.7.11	EDMA_SetChannelLink	181
12.7.12	EDMA_SetBandWidth	182
12.7.13	EDMA_SetModulo	182
12.7.14	EDMA_EnableAsyncRequest	183
12.7.15	EDMA_EnableAutoStopRequest	183
12.7.16	EDMA_EnableChannelInterrupts	183
12.7.17	EDMA_DisableChannelInterrupts	183
12.7.18	EDMA_SetMajorOffsetConfig	184
12.7.19	EDMA_TcdReset	184
12.7.20	EDMA_TcdSetTransferConfig	184
12.7.21	EDMA_TcdSetMinorOffsetConfig	185
12.7.22	EDMA_TcdSetChannelLink	185
12.7.23	EDMA_TcdSetBandWidth	186
12.7.24	EDMA_TcdSetModulo	186
12.7.25	EDMA_TcdEnableAutoStopRequest	187
12.7.26	EDMA_TcdEnableInterrupts	188
12.7.27	EDMA_TcdDisableInterrupts	188
12.7.28	EDMA_TcdSetMajorOffsetConfig	188

Section No.	Title	Page No.
12.7.29	<code>EDMA_EnableChannelRequest</code>	188
12.7.30	<code>EDMA_DisableChannelRequest</code>	189
12.7.31	<code>EDMA_TriggerChannelStart</code>	189
12.7.32	<code>EDMA_GetRemainingMajorLoopCount</code>	189
12.7.33	<code>EDMA_GetErrorStatusFlags</code>	190
12.7.34	<code>EDMA_GetChannelStatusFlags</code>	190
12.7.35	<code>EDMA_ClearChannelStatusFlags</code>	190
12.7.36	<code>EDMA_CreateHandle</code>	191
12.7.37	<code>EDMA_InstallTCDMemory</code>	191
12.7.38	<code>EDMA_SetCallback</code>	191
12.7.39	<code>EDMA_PreparesTransferConfig</code>	192
12.7.40	<code>EDMA_PreparesTransfer</code>	192
12.7.41	<code>EDMA_SubmitTransfer</code>	193
12.7.42	<code>EDMA_StartTransfer</code>	194
12.7.43	<code>EDMA_StopTransfer</code>	195
12.7.44	<code>EDMA_AbortTransfer</code>	195
12.7.45	<code>EDMA_GetUnusedTCDNumber</code>	195
12.7.46	<code>EDMA_GetNextTCDAddress</code>	195
12.7.47	<code>EDMA_HandleIRQ</code>	196

Chapter 13 EWM: External Watchdog Monitor Driver

13.1	Overview	197
13.2	Typical use case	197
13.3	Data Structure Documentation	198
13.3.1	<code>struct ewm_config_t</code>	198
13.4	Macro Definition Documentation	198
13.4.1	<code>FSL_EWM_DRIVER_VERSION</code>	198
13.5	Enumeration Type Documentation	198
13.5.1	<code>_ewm_interrupt_enable_t</code>	198
13.5.2	<code>_ewm_status_flags_t</code>	198
13.6	Function Documentation	199
13.6.1	<code>EWM_Init</code>	199
13.6.2	<code>EWM_Deinit</code>	199
13.6.3	<code>EWM_GetDefaultConfig</code>	199
13.6.4	<code>EWM_EnableInterrupts</code>	200
13.6.5	<code>EWM_DisableInterrupts</code>	200
13.6.6	<code>EWM_GetStatusFlags</code>	200
13.6.7	<code>EWM_Refresh</code>	201

Section No.	Title	Page No.
Chapter 14 C90TFS Flash Driver		
14.1	Overview	202
14.2	Ftftx FLASH Driver	203
14.2.1	Overview	203
14.2.2	Data Structure Documentation	205
14.2.3	Macro Definition Documentation	206
14.2.4	Enumeration Type Documentation	206
14.2.5	Function Documentation	207
14.3	Ftftx CACHE Driver	223
14.3.1	Overview	223
14.3.2	Data Structure Documentation	223
14.3.3	Enumeration Type Documentation	224
14.3.4	Function Documentation	224
14.4	Ftftx FLEXNVM Driver	227
14.4.1	Overview	227
14.4.2	Data Structure Documentation	229
14.4.3	Enumeration Type Documentation	229
14.4.4	Function Documentation	229
14.5	ftfx feature	243
14.5.1	Overview	243
14.5.2	Macro Definition Documentation	243
14.5.3	ftfx adapter	244
14.6	ftfx controller	245
14.6.1	Overview	245
14.6.2	Data Structure Documentation	248
14.6.3	Macro Definition Documentation	250
14.6.4	Enumeration Type Documentation	250
14.6.5	Function Documentation	252
14.6.6	ftfx utilities	264
Chapter 15 FlexBus: External Bus Interface Driver		
15.1	Overview	265
15.2	FlexBus functional operation	265
15.3	Typical use case and example	265
15.4	Data Structure Documentation	267
15.4.1	struct flexbus_config_t	267

Section No.	Title	Page No.
15.5 Macro Definition Documentation		268
15.5.1 FSL_FLEXBUS_DRIVER_VERSION		268
15.6 Enumeration Type Documentation		268
15.6.1 flexbus_port_size_t		268
15.6.2 flexbus_write_address_hold_t		268
15.6.3 flexbus_read_address_hold_t		268
15.6.4 flexbus_address_setup_t		269
15.6.5 flexbus_bytelane_shift_t		269
15.6.6 flexbus_multiplex_group1_t		269
15.6.7 flexbus_multiplex_group2_t		269
15.6.8 flexbus_multiplex_group3_t		270
15.6.9 flexbus_multiplex_group4_t		270
15.6.10 flexbus_multiplex_group5_t		270
15.7 Function Documentation		270
15.7.1 FLEXBUS_Init		270
15.7.2 FLEXBUS_Deinit		271
15.7.3 FLEXBUS_GetDefaultConfig		271

Chapter 16 FTM: FlexTimer Driver

16.1 Overview		272
16.2 Function groups		272
16.2.1 Initialization and deinitialization		272
16.2.2 PWM Operations		272
16.2.3 Input capture operations		272
16.2.4 Output compare operations		273
16.2.5 Quad decode		273
16.2.6 Fault operation		273
16.3 Register Update		273
16.4 Typical use case		273
16.4.1 PWM output		274
16.5 Data Structure Documentation		280
16.5.1 struct ftm_chnl_pwm_signal_param_t		280
16.5.2 struct ftm_chnl_pwm_config_param_t		281
16.5.3 struct ftm_dual_edge_capture_param_t		282
16.5.4 struct ftm_phase_params_t		282
16.5.5 struct ftm_fault_param_t		282
16.5.6 struct ftm_config_t		283
16.6 Macro Definition Documentation		284

Section No.	Title	Page No.
16.6.1	FSL_FTM_DRIVER_VERSION	284
16.7	Enumeration Type Documentation	284
16.7.1	ftm_chnl_t	284
16.7.2	ftm_fault_input_t	284
16.7.3	ftm_pwm_mode_t	284
16.7.4	ftm_pwm_level_select_t	285
16.7.5	ftm_output_compare_mode_t	285
16.7.6	ftm_input_capture_edge_t	285
16.7.7	ftm_dual_edge_capture_mode_t	285
16.7.8	ftm_quad_decode_mode_t	285
16.7.9	ftm_phase_polarity_t	286
16.7.10	ftm_deadtime_prescale_t	286
16.7.11	ftm_clock_source_t	286
16.7.12	ftm_clock_prescale_t	286
16.7.13	ftm_bdm_mode_t	286
16.7.14	ftm_fault_mode_t	287
16.7.15	ftm_external_trigger_t	287
16.7.16	ftm_pwm_sync_method_t	287
16.7.17	ftm_reload_point_t	288
16.7.18	ftm_interrupt_enable_t	288
16.7.19	ftm_status_flags_t	288
16.7.20	anonymous enum	289
16.8	Function Documentation	289
16.8.1	FTM_Init	289
16.8.2	FTM_Deinit	290
16.8.3	FTM_GetDefaultConfig	290
16.8.4	FTM_CalculateCounterClkDiv	290
16.8.5	FTM_SetupPwm	291
16.8.6	FTM_UpdatePwmDutyCycle	292
16.8.7	FTM_UpdateChnlEdgeLevelSelect	292
16.8.8	FTM_SetupPwmMode	293
16.8.9	FTM_SetupInputCapture	293
16.8.10	FTM_SetupOutputCompare	294
16.8.11	FTM_SetupDualEdgeCapture	294
16.8.12	FTM_SetupFaultInput	295
16.8.13	FTM_EnableInterrupts	295
16.8.14	FTM_DisableInterrupts	295
16.8.15	FTM_GetEnabledInterrupts	295
16.8.16	FTM_GetStatusFlags	296
16.8.17	FTM_ClearStatusFlags	296
16.8.18	FTM_SetTimerPeriod	296
16.8.19	FTM_GetCurrentTimerCount	297
16.8.20	FTM_GetInputCaptureValue	297

Section No.	Title	Page No.
16.8.21	FTM_StartTimer	298
16.8.22	FTM_StopTimer	298
16.8.23	FTM_SetSoftwareCtrlEnable	298
16.8.24	FTM_SetSoftwareCtrlVal	298
16.8.25	FTM_SetGlobalTimeBaseOutputEnable	299
16.8.26	FTM_SetOutputMask	299
16.8.27	FTM_SetFaultControlEnable	299
16.8.28	FTM_SetDeadTimeEnable	300
16.8.29	FTM_SetComplementaryEnable	300
16.8.30	FTM_SetInvertEnable	300
16.8.31	FTM_SetupQuadDecode	301
16.8.32	FTM_GetQuadDecoderFlags	301
16.8.33	FTM_SetQuadDecoderModuloValue	301
16.8.34	FTM_GetQuadDecoderCounterValue	302
16.8.35	FTM_ClearQuadDecoderCounterValue	302
16.8.36	FTM_SetSoftwareTrigger	302
16.8.37	FTM_SetWriteProtection	302
16.8.38	FTM_EnableDmaTransfer	303

Chapter 17 GPIO: General-Purpose Input/Output Driver

17.1	Overview	304
17.2	Data Structure Documentation	304
17.2.1	struct gpio_pin_config_t	304
17.3	Macro Definition Documentation	305
17.3.1	FSL_GPIO_DRIVER_VERSION	305
17.4	Enumeration Type Documentation	305
17.4.1	gpio_pin_direction_t	305
17.5	GPIO Driver	306
17.5.1	Overview	306
17.5.2	Typical use case	306
17.5.3	Function Documentation	307
17.6	FGPIO Driver	310
17.6.1	Typical use case	310

Chapter 18 I2C: Inter-Integrated Circuit Driver

18.1	Overview	311
18.2	I2C Driver	312
18.2.1	Overview	312

Section No.	Title	Page No.
18.2.2	Typical use case	312
18.2.3	Data Structure Documentation	317
18.2.4	Macro Definition Documentation	321
18.2.5	Typedef Documentation	321
18.2.6	Enumeration Type Documentation	321
18.2.7	Function Documentation	323
18.3	I2C eDMA Driver	337
18.3.1	Overview	337
18.3.2	Data Structure Documentation	337
18.3.3	Macro Definition Documentation	338
18.3.4	Typedef Documentation	338
18.3.5	Function Documentation	338
18.4	I2C FreeRTOS Driver	341
18.4.1	Overview	341
18.4.2	Macro Definition Documentation	341
18.4.3	Function Documentation	341
18.5	I2C CMSIS Driver	344
18.5.1	I2C CMSIS Driver	344

Chapter 19 LLWU: Low-Leakage Wakeup Unit Driver

19.1	Overview	346
19.2	External wakeup pins configurations	346
19.3	Internal wakeup modules configurations	346
19.4	Digital pin filter for external wakeup pin configurations	346
19.5	Data Structure Documentation	347
19.5.1	struct llwu_external_pin_filter_mode_t	347
19.6	Macro Definition Documentation	347
19.6.1	FSL_LLWU_DRIVER_VERSION	347
19.7	Enumeration Type Documentation	347
19.7.1	llwu_external_pin_mode_t	347
19.7.2	llwu_pin_filter_mode_t	348
19.8	Function Documentation	348
19.8.1	LLWU_SetExternalWakePinMode	348
19.8.2	LLWU_GetExternalWakePinFlag	348
19.8.3	LLWU_ClearExternalWakePinFlag	349

Section No.	Title	Page No.
19.8.4	<code>LLWU_EnableInternalModuleInterruptWakeup</code>	350
19.8.5	<code>LLWU_GetInternalWakeupModuleFlag</code>	350
19.8.6	<code>LLWU_SetPinFilterMode</code>	350
19.8.7	<code>LLWU_GetPinFilterFlag</code>	351
19.8.8	<code>LLWU_ClearPinFilterFlag</code>	351

Chapter 20 LPTMR: Low-Power Timer

20.1	Overview	352
20.2	Function groups	352
20.2.1	Initialization and deinitialization	352
20.2.2	Timer period Operations	352
20.2.3	Start and Stop timer operations	352
20.2.4	Status	353
20.2.5	Interrupt	353
20.3	Typical use case	353
20.3.1	LPTMR tick example	353
20.4	Data Structure Documentation	355
20.4.1	<code>struct lptmr_config_t</code>	355
20.5	Enumeration Type Documentation	356
20.5.1	<code>lptmr_pin_select_t</code>	356
20.5.2	<code>lptmr_pin_polarity_t</code>	356
20.5.3	<code>lptmr_timer_mode_t</code>	356
20.5.4	<code>lptmr_prescaler_glitch_value_t</code>	356
20.5.5	<code>lptmr_prescaler_clock_select_t</code>	357
20.5.6	<code>lptmr_interrupt_enable_t</code>	357
20.5.7	<code>lptmr_status_flags_t</code>	357
20.6	Function Documentation	357
20.6.1	<code>LPTMR_Init</code>	357
20.6.2	<code>LPTMR_Deinit</code>	358
20.6.3	<code>LPTMR_GetDefaultConfig</code>	358
20.6.4	<code>LPTMR_EnableInterrupts</code>	358
20.6.5	<code>LPTMR_DisableInterrupts</code>	358
20.6.6	<code>LPTMR_GetEnabledInterrupts</code>	359
20.6.7	<code>LPTMR_GetStatusFlags</code>	359
20.6.8	<code>LPTMR_ClearStatusFlags</code>	359
20.6.9	<code>LPTMR_SetTimerPeriod</code>	360
20.6.10	<code>LPTMR_GetCurrentTimerCount</code>	360
20.6.11	<code>LPTMR_StartTimer</code>	360
20.6.12	<code>LPTMR_StopTimer</code>	361

Section No.	Title	Page No.
Chapter 21 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver		
21.1	Overview	362
21.2	LPUART Driver	363
21.2.1	Overview	363
21.2.2	Typical use case	363
21.2.3	Data Structure Documentation	368
21.2.4	Macro Definition Documentation	370
21.2.5	Typedef Documentation	370
21.2.6	Enumeration Type Documentation	371
21.2.7	Function Documentation	373
21.3	LPUART eDMA Driver	391
21.3.1	Overview	391
21.3.2	Data Structure Documentation	392
21.3.3	Macro Definition Documentation	392
21.3.4	Typedef Documentation	392
21.3.5	Function Documentation	393
21.4	LPUART FreeRTOS Driver	397
21.4.1	Overview	397
21.4.2	Data Structure Documentation	397
21.4.3	Macro Definition Documentation	398
21.4.4	Function Documentation	398
21.5	LPUART CMSIS Driver	401
21.5.1	Function groups	401
Chapter 22 PDB: Programmable Delay Block		
22.1	Overview	403
22.2	Typical use case	403
22.2.1	Working as basic PDB counter with a PDB interrupt.	403
22.2.2	Working with an additional trigger. The ADC trigger is used as an example.	403
22.3	Data Structure Documentation	407
22.3.1	struct pdb_config_t	407
22.3.2	struct pdb_adc_pretrigger_config_t	408
22.3.3	struct pdb_dac_trigger_config_t	408
22.4	Macro Definition Documentation	409
22.4.1	FSL_PDB_DRIVER_VERSION	409
22.5	Enumeration Type Documentation	409

Section No.	Title	Page No.
22.5.1	<code>_pdb_status_flags</code>	409
22.5.2	<code>_pdb_adc_pretrigger_flags</code>	409
22.5.3	<code>_pdb_interrupt_enable</code>	409
22.5.4	<code>pdb_load_value_mode_t</code>	409
22.5.5	<code>pdb_prescaler_divider_t</code>	410
22.5.6	<code>pdb_divider_multiplication_factor_t</code>	410
22.5.7	<code>pdb_trigger_input_source_t</code>	410
22.5.8	<code>pdb_adc_trigger_channel_t</code>	411
22.5.9	<code>pdb_adc_pretrigger_t</code>	411
22.5.10	<code>pdb_dac_trigger_channel_t</code>	412
22.5.11	<code>pdb_pulse_out_trigger_channel_t</code>	412
22.5.12	<code>pdb_pulse_out_channel_mask_t</code>	412
22.6	Function Documentation	412
22.6.1	<code>PDB_Init</code>	412
22.6.2	<code>PDB_Deinit</code>	413
22.6.3	<code>PDB_GetDefaultConfig</code>	413
22.6.4	<code>PDB_Enable</code>	413
22.6.5	<code>PDB_DoSoftwareTrigger</code>	413
22.6.6	<code>PDB_DoLoadValues</code>	414
22.6.7	<code>PDB_EnableDMA</code>	414
22.6.8	<code>PDB_EnableInterrupts</code>	414
22.6.9	<code>PDB_DisableInterrupts</code>	414
22.6.10	<code>PDB_GetStatusFlags</code>	415
22.6.11	<code>PDB_ClearStatusFlags</code>	415
22.6.12	<code>PDB_SetModulusValue</code>	415
22.6.13	<code>PDB_GetCounterValue</code>	415
22.6.14	<code>PDB_SetCounterDelayValue</code>	416
22.6.15	<code>PDB_SetADCPreTriggerConfig</code>	416
22.6.16	<code>PDB_SetADCPreTriggerDelayValue</code>	416
22.6.17	<code>PDB_GetADCPreTriggerStatusFlags</code>	417
22.6.18	<code>PDB_ClearADCPreTriggerStatusFlags</code>	417
22.6.19	<code>PDB_SetDACTriggerConfig</code>	417
22.6.20	<code>PDB_SetDACTriggerIntervalValue</code>	418
22.6.21	<code>PDB_EnablePulseOutTrigger</code>	418
22.6.22	<code>PDB_SetPulseOutTriggerDelayValue</code>	418

Chapter 23 PIT: Periodic Interrupt Timer

23.1	Overview	420
23.2	Function groups	420
23.2.1	<code>Initialization and deinitialization</code>	420
23.2.2	<code>Timer period Operations</code>	420
23.2.3	<code>Start and Stop timer operations</code>	420

Section No.	Title	Page No.
23.2.4	Status	421
23.2.5	Interrupt	421
23.3	Typical use case	421
23.3.1	PIT tick example	421
23.4	Data Structure Documentation	422
23.4.1	struct pit_config_t	422
23.5	Enumeration Type Documentation	422
23.5.1	pit_chnl_t	422
23.5.2	pit_interrupt_enable_t	423
23.5.3	pit_status_flags_t	423
23.6	Function Documentation	423
23.6.1	PIT_Init	423
23.6.2	PIT_Deinit	423
23.6.3	PIT_GetDefaultConfig	424
23.6.4	PIT_SetTimerChainMode	424
23.6.5	PIT_EnableInterrupts	424
23.6.6	PIT_DisableInterrupts	425
23.6.7	PIT_GetEnabledInterrupts	425
23.6.8	PIT_GetStatusFlags	425
23.6.9	PIT_ClearStatusFlags	426
23.6.10	PIT_SetTimerPeriod	426
23.6.11	PIT_GetCurrentTimerCount	427
23.6.12	PIT_StartTimer	427
23.6.13	PIT_StopTimer	427

Chapter 24 PMC: Power Management Controller

24.1	Overview	429
24.2	Data Structure Documentation	430
24.2.1	struct pmc_low_volt_detect_config_t	430
24.2.2	struct pmc_low_volt_warning_config_t	430
24.2.3	struct pmc_bandgap_buffer_config_t	430
24.3	Macro Definition Documentation	431
24.3.1	FSL_PMC_DRIVER_VERSION	431
24.4	Enumeration Type Documentation	431
24.4.1	pmc_low_volt_detect_volt_select_t	431
24.4.2	pmc_low_volt_warning_volt_select_t	431
24.5	Function Documentation	431

Section No.	Title	Page No.
24.5.1	PMC_ConfigureLowVoltDetect	431
24.5.2	PMC_GetLowVoltDetectFlag	432
24.5.3	PMC_ClearLowVoltDetectFlag	432
24.5.4	PMC_ConfigureLowVoltWarning	432
24.5.5	PMC_GetLowVoltWarningFlag	433
24.5.6	PMC_ClearLowVoltWarningFlag	433
24.5.7	PMC_ConfigureBandgapBuffer	433
24.5.8	PMC_GetPeriphIOIsolationFlag	434
24.5.9	PMC_ClearPeriphIOIsolationFlag	434
24.5.10	PMC_IsRegulatorInRunRegulation	434

Chapter 25 PORT: Port Control and Interrupts

25.1	Overview	436
25.2	Data Structure Documentation	438
25.2.1	struct port_digital_filter_config_t	438
25.2.2	struct port_pin_config_t	438
25.3	Macro Definition Documentation	439
25.3.1	FSL_PORT_DRIVER_VERSION	439
25.4	Enumeration Type Documentation	439
25.4.1	_port_pull	439
25.4.2	_port_slew_rate	439
25.4.3	_port_open_drain_enable	439
25.4.4	_port_passive_filter_enable	439
25.4.5	_port_drive_strength	440
25.4.6	_port_lock_register	440
25.4.7	port_mux_t	440
25.4.8	port_interrupt_t	440
25.4.9	port_digital_filter_clock_source_t	441
25.5	Function Documentation	441
25.5.1	PORT_SetPinConfig	441
25.5.2	PORT_SetMultiplePinsConfig	442
25.5.3	PORT_SetPinMux	442
25.5.4	PORT_EnablePinsDigitalFilter	443
25.5.5	PORT_SetDigitalFilterConfig	443
25.5.6	PORT_SetPinInterruptConfig	444
25.5.7	PORT_SetPinDriveStrength	444
25.5.8	PORT_GetPinsInterruptFlags	445
25.5.9	PORT_ClearPinsInterruptFlags	445

Section No.	Title	Page No.
Chapter 26 RCM: Reset Control Module Driver		
26.1 Overview	446
26.2 Data Structure Documentation	447
26.2.1 struct rcm_reset_pin_filter_config_t	447
26.3 Macro Definition Documentation	447
26.3.1 FSL_RCM_DRIVER_VERSION	447
26.4 Enumeration Type Documentation	447
26.4.1 rcm_reset_source_t	447
26.4.2 rcm_run_wait_filter_mode_t	448
26.5 Function Documentation	448
26.5.1 RCM_GetPreviousResetSources	448
26.5.2 RCM_GetStickyResetSources	449
26.5.3 RCM_ClearStickyResetSources	449
26.5.4 RCM_ConfigureResetPinFilter	450
26.5.5 RCM_GetEasyPortModePinStatus	451
Chapter 27 RNGA: Random Number Generator Accelerator Driver		
27.1 Overview	452
27.2 RNGA Initialization	452
27.3 Get random data from RNGA	452
27.4 RNGA Set/Get Working Mode	452
27.5 Seed RNGA	452
27.6 Macro Definition Documentation	453
27.6.1 FSL_RNGA_DRIVER_VERSION	453
27.7 Enumeration Type Documentation	453
27.7.1 rnga_mode_t	454
27.8 Function Documentation	454
27.8.1 RNGA_Init	454
27.8.2 RNGA_Deinit	454
27.8.3 RNGA_GetRandomData	454
27.8.4 RNGA_Seed	455
27.8.5 RNGA_SetMode	456
27.8.6 RNGA_GetMode	456

Section No.	Title	Page No.
Chapter 28 RTC: Real Time Clock		
28.1	Overview	457
28.2	Function groups	457
28.2.1	Initialization and deinitialization	457
28.2.2	Set & Get Datetime	457
28.2.3	Set & Get Alarm	457
28.2.4	Start & Stop timer	457
28.2.5	Status	458
28.2.6	Interrupt	458
28.2.7	RTC Oscillator	458
28.2.8	Monotonic Counter	458
28.3	Typical use case	458
28.3.1	RTC tick example	458
28.4	Data Structure Documentation	460
28.4.1	struct rtc_datetime_t	460
28.4.2	struct rtc_config_t	461
28.5	Enumeration Type Documentation	461
28.5.1	rtc_interrupt_enable_t	461
28.5.2	rtc_status_flags_t	461
28.5.3	rtc_osc_cap_load_t	462
28.6	Function Documentation	462
28.6.1	RTC_Init	462
28.6.2	RTC_Deinit	462
28.6.3	RTC_GetDefaultConfig	462
28.6.4	RTC_SetDatetime	463
28.6.5	RTC_GetDatetime	463
28.6.6	RTC_SetAlarm	463
28.6.7	RTC_GetAlarm	464
28.6.8	RTC_EnableInterrupts	464
28.6.9	RTC_DisableInterrupts	464
28.6.10	RTC_GetEnabledInterrupts	464
28.6.11	RTC_GetStatusFlags	465
28.6.12	RTC_ClearStatusFlags	465
28.6.13	RTC_SetClockSource	465
28.6.14	RTC_StartTimer	466
28.6.15	RTC_StopTimer	467
28.6.16	RTC_SetOscCapLoad	467
28.6.17	RTC_Reset	467
28.6.18	RTC_EnableWakeUpPin	467

Section No.	Title	Page No.
Chapter 29 SAI: Serial Audio Interface		
29.1	Overview	469
29.2	Typical configurations	469
29.3	Typical use case	470
29.3.1	SAI Send/receive using an interrupt method	470
29.3.2	SAI Send/receive using a DMA method	470
29.4	SAI Driver	471
29.4.1	Overview	471
29.4.2	Data Structure Documentation	479
29.4.3	Macro Definition Documentation	483
29.4.4	Enumeration Type Documentation	483
29.4.5	Function Documentation	487
29.5	SAI EDMA Driver	516
29.5.1	Overview	516
29.5.2	Data Structure Documentation	517
29.5.3	Function Documentation	518
Chapter 30 SIM: System Integration Module Driver		
30.1	Overview	529
30.2	Data Structure Documentation	529
30.2.1	struct sim_uid_t	529
30.3	Enumeration Type Documentation	530
30.3.1	_sim_usb_volt_reg_enable_mode	530
30.3.2	_sim_flash_mode	530
30.4	Function Documentation	530
30.4.1	SIM_SetUsbVoltRegulatorEnableMode	530
30.4.2	SIM_GetUniqueId	531
30.4.3	SIM_SetFlashMode	531
Chapter 31 SMC: System Mode Controller Driver		
31.1	Overview	532
31.2	Typical use case	532
31.2.1	Enter wait or stop modes	532
31.3	Data Structure Documentation	534

Section No.	Title	Page No.
31.3.1	struct smc_power_mode_lls_config_t	535
31.3.2	struct smc_power_mode_vlls_config_t	535
31.4	Enumeration Type Documentation	535
31.4.1	smc_power_mode_protection_t	535
31.4.2	smc_power_state_t	535
31.4.3	smc_run_mode_t	536
31.4.4	smc_stop_mode_t	536
31.4.5	smc_stop_submode_t	536
31.4.6	smc_partial_stop_option_t	536
31.4.7	anonymous enum	536
31.5	Function Documentation	537
31.5.1	SMC_SetPowerModeProtection	537
31.5.2	SMC_GetPowerModeState	537
31.5.3	SMC_PreEnterStopModes	537
31.5.4	SMC_PostExitStopModes	538
31.5.5	SMC_PreEnterWaitModes	538
31.5.6	SMC_PostExitWaitModes	538
31.5.7	SMC_SetPowerModeRun	538
31.5.8	SMC_SetPowerModeHsrun	538
31.5.9	SMC_SetPowerModeWait	538
31.5.10	SMC_SetPowerModeStop	539
31.5.11	SMC_SetPowerModeVlpr	539
31.5.12	SMC_SetPowerModeVlpw	539
31.5.13	SMC_SetPowerModeVlps	540
31.5.14	SMC_SetPowerModeLls	540
31.5.15	SMC_SetPowerModeVlls	540

Chapter 32 UART: Universal Asynchronous Receiver/Transmitter Driver

32.1	Overview	542
32.2	UART Driver	543
32.2.1	Overview	543
32.2.2	Typical use case	543
32.2.3	Data Structure Documentation	549
32.2.4	Macro Definition Documentation	551
32.2.5	Typedef Documentation	551
32.2.6	Enumeration Type Documentation	551
32.2.7	Function Documentation	553
32.2.8	Variable Documentation	569
32.3	UART eDMA Driver	570
32.3.1	Overview	570

Section No.	Title	Page No.
32.3.2	Data Structure Documentation	571
32.3.3	Macro Definition Documentation	571
32.3.4	Typedef Documentation	571
32.3.5	Function Documentation	572
32.4	UART FreeRTOS Driver	577
32.4.1	Overview	577
32.4.2	Data Structure Documentation	577
32.4.3	Macro Definition Documentation	578
32.4.4	Function Documentation	578
32.5	UART CMSIS Driver	580
32.5.1	UART CMSIS Driver	580
Chapter 33 VREF: Voltage Reference Driver		
33.1	Overview	582
33.2	VREF functional Operation	582
33.3	Typical use case and example	582
33.4	Data Structure Documentation	583
33.4.1	struct vref_config_t	583
33.5	Macro Definition Documentation	583
33.5.1	FSL_VREF_DRIVER_VERSION	583
33.6	Enumeration Type Documentation	583
33.6.1	vref_buffer_mode_t	583
33.7	Function Documentation	583
33.7.1	VREF_Init	583
33.7.2	VREF_Deinit	584
33.7.3	VREF_GetDefaultConfig	584
33.7.4	VREF_SetTrimVal	584
33.7.5	VREF_GetTrimVal	585
Chapter 34 WDOG: Watchdog Timer Driver		
34.1	Overview	586
34.2	Typical use case	586
34.3	Data Structure Documentation	588
34.3.1	struct wdog_work_mode_t	588

Section No.	Title	Page No.
34.3.2	struct wdog_config_t	588
34.3.3	struct wdog_test_config_t	589
34.4	Macro Definition Documentation	589
34.4.1	FSL_WDOG_DRIVER_VERSION	589
34.5	Enumeration Type Documentation	589
34.5.1	wdog_clock_source_t	589
34.5.2	wdog_clock_prescaler_t	589
34.5.3	wdog_test_mode_t	590
34.5.4	wdog_tested_byte_t	590
34.5.5	_wdog_interrupt_enable_t	590
34.5.6	_wdog_status_flags_t	590
34.6	Function Documentation	590
34.6.1	WDOG_GetDefaultConfig	590
34.6.2	WDOG_Init	591
34.6.3	WDOG_Deinit	591
34.6.4	WDOG_SetTestModeConfig	592
34.6.5	WDOG_Enable	592
34.6.6	WDOG_Disable	592
34.6.7	WDOG_EnableInterrupts	593
34.6.8	WDOG_DisableInterrupts	593
34.6.9	WDOG_GetStatusFlags	593
34.6.10	WDOG_ClearStatusFlags	594
34.6.11	WDOG_SetTimeoutValue	594
34.6.12	WDOG_SetWindowValue	595
34.6.13	WDOG_Unlock	595
34.6.14	WDOG_Refresh	595
34.6.15	WDOG_GetResetCount	596
34.6.16	WDOG_ClearResetCount	597
Chapter 35 Debug Console		
35.1	Overview	598
35.2	Function groups	598
35.2.1	Initialization	598
35.2.2	Advanced Feature	599
35.2.3	SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART	603
35.3	Typical use case	604
35.4	Macro Definition Documentation	606
35.4.1	DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN	606
35.4.2	DEBUGCONSOLE_REDIRECT_TO_SDK	606

Section No.	Title	Page No.
35.4.3	DEBUGCONSOLE_DISABLE	606
35.4.4	SDK_DEBUGCONSOLE	606
35.4.5	PRINTF	606
35.5	Function Documentation	606
35.5.1	DbgConsole_Init	606
35.5.2	DbgConsole_Deinit	607
35.5.3	DbgConsole_EnterLowpower	607
35.5.4	DbgConsole_ExitLowpower	608
35.5.5	DbgConsole_Printf	608
35.5.6	DbgConsole_Vprintf	608
35.5.7	DbgConsole_Putchar	608
35.5.8	DbgConsole_Scanf	609
35.5.9	DbgConsole_Getchar	609
35.5.10	DbgConsole_BlockingPrintf	610
35.5.11	DbgConsole_BlockingVprintf	610
35.5.12	DbgConsole_Flush	610
35.5.13	StrFormatPrintf	611
35.5.14	StrFormatScanf	611
35.6	Semihosting	612
35.6.1	Guide Semihosting for IAR	612
35.6.2	Guide Semihosting for Keil µVision	612
35.6.3	Guide Semihosting for MCUXpresso IDE	613
35.6.4	Guide Semihosting for ARMGCC	613
35.7	SWO	616
35.7.1	Guide SWO for SDK	616
35.7.2	Guide SWO for Keil µVision	617
35.7.3	Guide SWO for MCUXpresso IDE	618
35.7.4	Guide SWO for ARMGCC	618
Chapter 36 Notification Framework		
36.1	Overview	619
36.2	Notifier Overview	619
36.3	Data Structure Documentation	621
36.3.1	struct notifier_notification_block_t	621
36.3.2	struct notifier_callback_config_t	622
36.3.3	struct notifier_handle_t	622
36.4	Typedef Documentation	623
36.4.1	notifier_user_config_t	623
36.4.2	notifier_user_function_t	623

Section No.	Title	Page No.
36.4.3	notifier_callback_t	624
36.5	Enumeration Type Documentation	624
36.5.1	_notifier_status	624
36.5.2	notifier_policy_t	625
36.5.3	notifier_notification_type_t	625
36.5.4	notifier_callback_type_t	625
36.6	Function Documentation	625
36.6.1	NOTIFIER_CreateHandle	626
36.6.2	NOTIFIER_SwitchConfig	627
36.6.3	NOTIFIER_GetErrorCallbackIndex	628
Chapter 37 Shell		
37.1	Overview	629
37.2	Function groups	629
37.2.1	Initialization	629
37.2.2	Advanced Feature	629
37.2.3	Shell Operation	629
37.3	Data Structure Documentation	631
37.3.1	struct shell_command_t	631
37.4	Macro Definition Documentation	632
37.4.1	SHELL_NON_BLOCKING_MODE	632
37.4.2	SHELL_AUTO_COMPLETE	632
37.4.3	SHELL_BUFFER_SIZE	632
37.4.4	SHELL_MAX_ARGS	632
37.4.5	SHELL_HISTORY_COUNT	632
37.4.6	SHELL_HANDLE_SIZE	632
37.4.7	SHELL_USE_COMMON_TASK	632
37.4.8	SHELL_TASK_PRIORITY	632
37.4.9	SHELL_TASK_STACK_SIZE	632
37.4.10	SHELL_HANDLE_DEFINE	633
37.4.11	SHELL_COMMAND_DEFINE	633
37.4.12	SHELL_COMMAND	634
37.5	Typedef Documentation	634
37.5.1	cmd_function_t	634
37.6	Enumeration Type Documentation	634
37.6.1	shell_status_t	634
37.7	Function Documentation	634

Section No.	Title	Page No.
37.7.1	SHELL_Init	634
37.7.2	SHELL_RegisterCommand	635
37.7.3	SHELL_UnregisterCommand	636
37.7.4	SHELL_Write	636
37.7.5	SHELL_Printf	636
37.7.6	SHELL_WriteSynchronization	637
37.7.7	SHELL_PrintfSynchronization	637
37.7.8	SHELL_ChangePrompt	638
37.7.9	SHELL_PrintPrompt	638
37.7.10	SHELL_Task	638
37.7.11	SHELL_checkRunningInIsr	639

Chapter 38 Cards: Secure Digital Card/Embedded MultiMedia Card/SDIO Card

38.1	Overview	640
38.2	SDMMC HOST Driver	641
38.3	SDMMC Common	642
38.3.1	Overview	642
38.3.2	Data Structure Documentation	661
38.3.3	Macro Definition Documentation	674
38.3.4	Enumeration Type Documentation	674
38.3.5	Function Documentation	691

Chapter 39 SPI based Secure Digital Card (SDSPI)

39.1	Overview	696
39.2	Data Structure Documentation	698
39.2.1	struct sdspi_host_t	698
39.2.2	struct sdspi_card_t	699
39.3	Macro Definition Documentation	699
39.3.1	FSL_SDSPI_DRIVER_VERSION	699
39.3.2	DSPI_DUMMY_DATA	699
39.3.3	SDSPI_CARD_CRC_PROTECTION_ENABLE	700
39.4	Enumeration Type Documentation	700
39.4.1	anonymous enum	700
39.4.2	anonymous enum	700
39.4.3	anonymous enum	701
39.4.4	anonymous enum	701
39.4.5	sdspi_cs_active_polarity_t	701
39.5	Function Documentation	701

Section No.	Title	Page No.
39.5.1	SDSPI_Init	701
39.5.2	SDSPI_Deinit	702
39.5.3	SDSPI_CheckReadOnly	702
39.5.4	SDSPI_ReadBlocks	703
39.5.5	SDSPI_WriteBlocks	703
39.5.6	SDSPI_SendCid	704
39.5.7	SDSPI_SendPreErase	705
39.5.8	SDSPI_EraseBlocks	705
39.5.9	SDSPI_SwitchToHighSpeed	706

Chapter 40 CODEC Driver

40.1	Overview	707
40.2	CODEC Common Driver	708
40.2.1	Overview	708
40.2.2	Data Structure Documentation	713
40.2.3	Macro Definition Documentation	714
40.2.4	Enumeration Type Documentation	714
40.2.5	Function Documentation	719
40.3	CODEC I2C Driver	723
40.3.1	Overview	723
40.3.2	Data Structure Documentation	724
40.3.3	Enumeration Type Documentation	724
40.3.4	Function Documentation	724
40.4	CS42888 Driver	727
40.4.1	Overview	727
40.4.2	Data Structure Documentation	729
40.4.3	Macro Definition Documentation	730
40.4.4	Enumeration Type Documentation	730
40.4.5	Function Documentation	731
40.4.6	CS42888 Adapter	737
40.5	DA7212 Driver	745
40.5.1	Overview	745
40.5.2	Data Structure Documentation	748
40.5.3	Macro Definition Documentation	749
40.5.4	Enumeration Type Documentation	749
40.5.5	Function Documentation	751
40.5.6	DA7212 Adapter	756
40.6	SGTL5000 Driver	764
40.6.1	Overview	764

Section No.	Title	Page No.
40.6.2	Data Structure Documentation	766
40.6.3	Macro Definition Documentation	767
40.6.4	Enumeration Type Documentation	767
40.6.5	Function Documentation	769
40.6.6	SGTL5000 Adapter	775
40.7	WM8960 Driver	783
40.7.1	Overview	783
40.7.2	Data Structure Documentation	786
40.7.3	Macro Definition Documentation	788
40.7.4	Enumeration Type Documentation	788
40.7.5	Function Documentation	790
40.7.6	WM8960 Adapter	797
40.8	WM8904 Driver	805
40.8.1	Overview	805
40.8.2	Data Structure Documentation	809
40.8.3	Macro Definition Documentation	810
40.8.4	Enumeration Type Documentation	810
40.8.5	Function Documentation	813
40.8.6	WM8904 Adapter	822

Chapter 41 Serial Manager

41.1	Overview	830
41.2	Data Structure Documentation	833
41.2.1	struct serial_manager_config_t	833
41.2.2	struct serial_manager_callback_message_t	833
41.3	Macro Definition Documentation	834
41.3.1	SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE	834
41.3.2	SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE	834
41.3.3	SERIAL_MANAGER_USE_COMMON_TASK	834
41.3.4	SERIAL_MANAGER_HANDLE_SIZE	834
41.3.5	SERIAL_MANAGER_HANDLE_DEFINE	834
41.3.6	SERIAL_MANAGER_WRITE_HANDLE_DEFINE	834
41.3.7	SERIAL_MANAGER_READ_HANDLE_DEFINE	835
41.3.8	SERIAL_MANAGER_TASK_PRIORITY	835
41.3.9	SERIAL_MANAGER_TASK_STACK_SIZE	835
41.4	Enumeration Type Documentation	835
41.4.1	serial_port_type_t	835
41.4.2	serial_manager_type_t	836
41.4.3	serial_manager_status_t	836

Section No.	Title	Page No.
41.5 Function Documentation		836
41.5.1 SerialManager_Init		836
41.5.2 SerialManager_Deinit		837
41.5.3 SerialManager_OpenWriteHandle		838
41.5.4 SerialManager_CloseWriteHandle		839
41.5.5 SerialManager_OpenReadHandle		839
41.5.6 SerialManager_CloseReadHandle		840
41.5.7 SerialManager_WriteBlocking		841
41.5.8 SerialManager_ReadBlocking		841
41.5.9 SerialManager_EnterLowpower		842
41.5.10 SerialManager_ExitLowpower		842
41.5.11 SerialManager_SetLowpowerCriticalCb		843
41.6 Serial Port Uart		844
41.6.1 Overview		844
41.6.2 Enumeration Type Documentation		844
41.7 Serial Port USB		845
41.7.1 Overview		845
41.7.2 Data Structure Documentation		845
41.7.3 Enumeration Type Documentation		846
41.7.4 USB Device Configuration		847
41.8 Serial Port SWO		848
41.8.1 Overview		848
41.8.2 Data Structure Documentation		848
41.8.3 Enumeration Type Documentation		848
41.8.4 CODEC Adapter		849

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 MCUXSDKRNN) 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.

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

MCUXpresso SDK Folder Structure

Chapter 2

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: nxp.com/SalesTermsandConditions.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EM-BRACE, GREENCHIP, HITAG, I2C BUS,ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4M-OBILE, 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, Arm7TD-MI, 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.

© 2021 NXP B.V.

Chapter 3

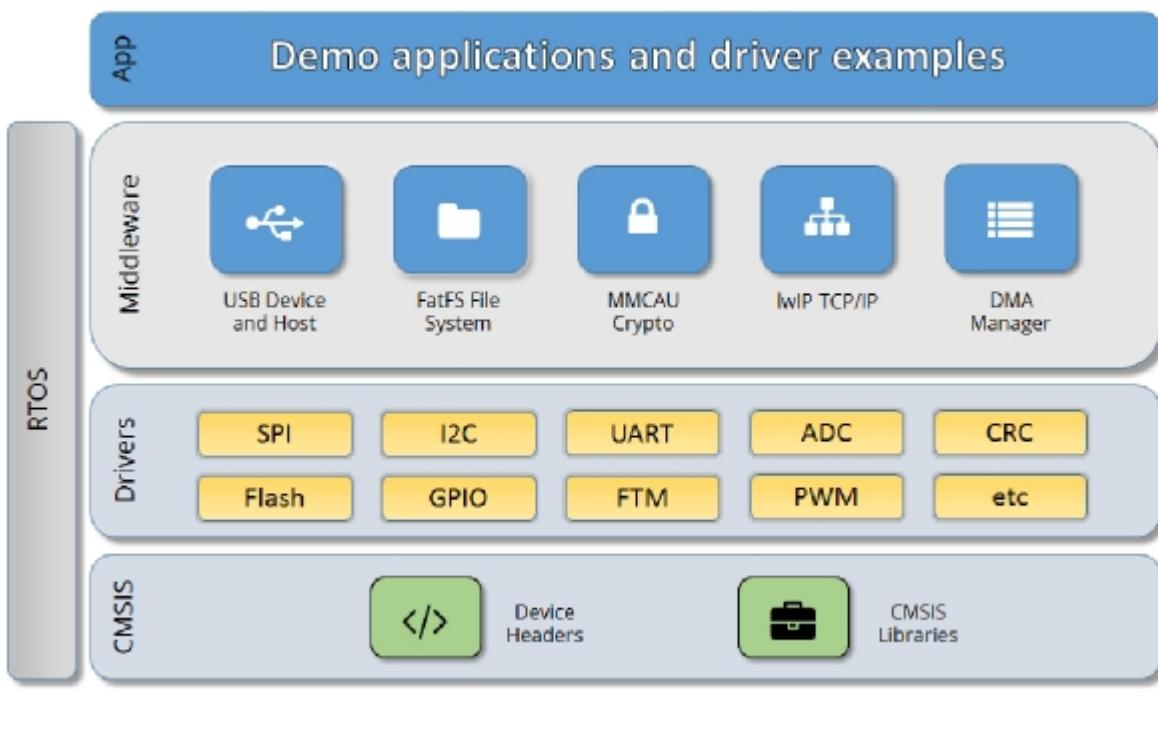
Architectural Overview

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



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 reimplementation 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 4

Clock Driver

4.1 Overview

The MCUXpresso SDK provides APIs for MCUXpresso SDK devices' clock operation.

The clock driver supports:

- Clock generator (PLL, FLL, and so on) configuration
- Clock mux and divider configuration
- Getting clock frequency

Modules

- Multipurpose Clock Generator (MCG)

Files

- file `fsl_clock.h`

Data Structures

- struct `sim_clock_config_t`
SIM configuration structure for clock setting. [More...](#)
- struct `oscer_config_t`
OSC configuration for OSCERCLK. [More...](#)
- struct `osc_config_t`
OSC Initialization Configuration Structure. [More...](#)
- struct `mcg_pll_config_t`
MCG PLL configuration. [More...](#)
- struct `mcg_config_t`
MCG mode change configuration structure. [More...](#)

Macros

- `#define MCG_CONFIG_CHECK_PARAM 0U`
Configures whether to check a parameter in a function.
- `#define FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL 0`
Configure whether driver controls clock.
- `#define MCG_INTERNAL_IRC_48M 48000000U`
IRC48M clock frequency in Hz.
- `#define DMAMUX_CLOCKS`
Clock ip name array for DMAMUX.
- `#define RTC_CLOCKS`
Clock ip name array for RTC.
- `#define SAI_CLOCKS`

- `#define PORT_CLOCKS`
Clock ip name array for PORT.
- `#define FLEXBUS_CLOCKS`
Clock ip name array for FLEXBUS.
- `#define EWM_CLOCKS`
Clock ip name array for EWM.
- `#define PIT_CLOCKS`
Clock ip name array for PIT.
- `#define DSPI_CLOCKS`
Clock ip name array for DSPI.
- `#define LPTMR_CLOCKS`
Clock ip name array for LPTMR.
- `#define FTM_CLOCKS`
Clock ip name array for FTM.
- `#define EDMA_CLOCKS`
Clock ip name array for EDMA.
- `#define LPUART_CLOCKS`
Clock ip name array for LPUART.
- `#define DAC_CLOCKS`
Clock ip name array for DAC.
- `#define ADC16_CLOCKS`
Clock ip name array for ADC16.
- `#define VREF_CLOCKS`
Clock ip name array for VREF.
- `#define UART_CLOCKS`
Clock ip name array for UART.
- `#define RNGA_CLOCKS`
Clock ip name array for RNGA.
- `#define CRC_CLOCKS`
Clock ip name array for CRC.
- `#define I2C_CLOCKS`
Clock ip name array for I2C.
- `#define FTF_CLOCKS`
Clock ip name array for FTF.
- `#define PDB_CLOCKS`
Clock ip name array for PDB.
- `#define CMP_CLOCKS`
Clock ip name array for CMP.
- `#define LPO_CLK_FREQ 1000U`
LPO clock frequency.
- `#define SYS_CLK kCLOCK_CoreSysClk`
Peripherals clock source definition.

Enumerations

- enum `clock_name_t` {

kCLOCK_CoreSysClk,

kCLOCK_PlatClk,

kCLOCK_BusClk,

kCLOCK_FlexBusClk,

kCLOCK_FlashClk,

kCLOCK_FastPeriphClk,

kCLOCK_PllFllSelClk,

kCLOCK_Er32kClk,

kCLOCK_Osc0ErClk,

kCLOCK_Osc1ErClk,

kCLOCK_Osc0ErClkUndiv,

kCLOCK_McgFixedFreqClk,

kCLOCK_McgInternalRefClk,

kCLOCK_McgFllClk,

kCLOCK_McgPll0Clk,

kCLOCK_McgPll1Clk,

kCLOCK_McgExtPllClk,

kCLOCK_McgPeriphClk,

kCLOCK_McgIrc48MClk,

kCLOCK_LpoClk }

Clock name used to get clock frequency.

- enum `clock_usb_src_t` {

kCLOCK_UsbSrcPll0 = SIM_SOPT2_USBSRC(1U) | SIM_SOPT2_PLLFLLSEL(1U),

kCLOCK_UsbSrcIrc48M = SIM_SOPT2_USBSRC(1U) | SIM_SOPT2_PLLFLLSEL(3U),

kCLOCK_UsbSrcExt = SIM_SOPT2_USBSRC(0U) }

USB clock source definition.

- enum `clock_ip_name_t`
 - Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.*

- enum `osc_mode_t` {

kOSC_ModeExt = 0U,

kOSC_ModeOscLowPower = MCG_C2_EREFST0_MASK,

kOSC_ModeOscHighGain }

OSC work mode.

- enum `_osc_cap_load` {

kOSC_Cap2P = OSC_CR_SC2P_MASK,

kOSC_Cap4P = OSC_CR_SC4P_MASK,

kOSC_Cap8P = OSC_CR_SC8P_MASK,

kOSC_Cap16P = OSC_CR_SC16P_MASK }

Oscillator capacitor load setting.

- enum `_oscer_enable_mode` {

kOSC_ErClkEnable = OSC_CR_ERCLKEN_MASK,

kOSC_ErClkEnableInStop = OSC_CR_EREFSTEN_MASK }

OSCERCLK enable mode.

- enum `mcg_fll_src_t` {

- ```
kMCG_FllSrcExternal,
kMCG_FllSrcInternal }
```

*MCG FLL reference clock source select.*
- enum `mcg_irc_mode_t` {
 

```
kMCG_IrcSlow,
kMCG_IrcFast }
```

*MCG internal reference clock select.*
- enum `mcg_dmx32_t` {
 

```
kMCG_Dmx32Default,
kMCG_Dmx32Fine }
```

*MCG DCO Maximum Frequency with 32.768 kHz Reference.*
- enum `mcg_drs_t` {
 

```
kMCG_DrsLow,
kMCG_DrsMid,
kMCG_DrsMidHigh,
kMCG_DrsHigh }
```

*MCG DCO range select.*
- enum `mcg_pll_ref_src_t` {
 

```
kMCG_PllRefOsc0,
kMCG_PllRefOsc1 }
```

*MCG PLL reference clock select.*
- enum `mcg_clkout_src_t` {
 

```
kMCG_ClkOutSrcOut,
kMCG_ClkOutSrcInternal,
kMCG_ClkOutSrcExternal }
```

*MCGOUT clock source.*
- enum `mcg_atm_select_t` {
 

```
kMCG_AtmSel32k,
kMCG_AtmSel4m }
```

*MCG Automatic Trim Machine Select.*
- enum `mcg_oscsel_t` {
 

```
kMCG_OscselOsc,
kMCG_OscselRtc,
kMCG_OscselIrc }
```

*MCG OSC Clock Select.*
- enum `mcg_pll_clk_select_t` { `kMCG_PllClkSelPll0` }
- enum `mcg_monitor_mode_t` {
 

```
kMCG_MonitorNone,
kMCG_MonitorInt,
kMCG_MonitorReset }
```

*MCG clock monitor mode.*
- enum { }

```

kStatus_MCG_ModeUnreachable = MAKE_STATUS(kStatusGroup_MCG, 0U),
kStatus_MCG_ModeInvalid = MAKE_STATUS(kStatusGroup_MCG, 1U),
kStatus_MCG_AtmBusClockInvalid = MAKE_STATUS(kStatusGroup_MCG, 2U),
kStatus_MCG_AtmDesiredFreqInvalid = MAKE_STATUS(kStatusGroup_MCG, 3U),
kStatus_MCG_AtmIrcUsed = MAKE_STATUS(kStatusGroup_MCG, 4U),
kStatus_MCG_AtmHardwareFail = MAKE_STATUS(kStatusGroup_MCG, 5U),
kStatus_MCG_SourceUsed = MAKE_STATUS(kStatusGroup_MCG, 6U) }

```

*MCG status.*

- enum {
   
kMCG\_Osc0LostFlag = (1U << 0U),
   
kMCG\_Osc0InitFlag = (1U << 1U),
   
kMCG\_RtcOscLostFlag = (1U << 4U),
   
kMCG\_Pll0LostFlag = (1U << 5U),
   
kMCG\_Pll0LockFlag = (1U << 6U) }

*MCG status flags.*

- enum {
   
kMCG\_IrclkEnable = MCG\_C1\_IRCLKEN\_MASK,
   
kMCG\_IrclkEnableInStop = MCG\_C1\_IREFSTEN\_MASK }
- MCG internal reference clock (MCGIRCLK) enable mode definition.*
- enum {
   
kMCG\_PllEnableIndependent = MCG\_C5\_PLLCLKEN0\_MASK,
   
kMCG\_PllEnableInStop = MCG\_C5\_PLLSTEN0\_MASK }
- MCG PLL clock enable mode definition.*

- enum `mcg_mode_t` {
   
kMCG\_ModeFEI = 0U,
   
kMCG\_ModeFBI,
   
kMCG\_ModeBLPI,
   
kMCG\_ModeFEE,
   
kMCG\_ModeFBE,
   
kMCG\_ModeBLPE,
   
kMCG\_ModePBE,
   
kMCG\_ModePEE,
   
kMCG\_ModeError }

*MCG mode definitions.*

## Functions

- static void `CLOCK_EnableClock (clock_ip_name_t name)`

*Enable the clock for specific IP.*
- static void `CLOCK_DisableClock (clock_ip_name_t name)`

*Disable the clock for specific IP.*
- static void `CLOCK_SetLpuartClock (uint32_t src)`

*Set LPUART clock source.*
- static void `CLOCK_SetEr32kClock (uint32_t src)`

*Set ERCLK32K source.*
- static void `CLOCK_SetTraceClock (uint32_t src)`

*Set debug trace clock source.*
- static void `CLOCK_SetPllFllSelClock (uint32_t src)`

- static void **CLOCK\_SetClkOutClock** (uint32\_t src)
  - Set CLKOUT source.*
- static void **CLOCK\_SetRtcClkOutClock** (uint32\_t src)
  - Set RTC\_CLKOUT source.*
- bool **CLOCK\_EnableUsbfs0Clock** (clock\_usb\_src\_t src, uint32\_t freq)
  - Enable USB FS clock.*
- static void **CLOCK\_DisableUsbfs0Clock** (void)
  - Disable USB FS clock.*
- static void **CLOCK\_SetOutDiv** (uint32\_t outdiv1, uint32\_t outdiv2, uint32\_t outdiv3, uint32\_t outdiv4)
  - System clock divider.*
- uint32\_t **CLOCK\_GetFreq** (clock\_name\_t clockName)
  - Gets the clock frequency for a specific clock name.*
- uint32\_t **CLOCK\_GetCoreSysClkFreq** (void)
  - Get the core clock or system clock frequency.*
- uint32\_t **CLOCK\_GetPlatClkFreq** (void)
  - Get the platform clock frequency.*
- uint32\_t **CLOCK\_GetBusClkFreq** (void)
  - Get the bus clock frequency.*
- uint32\_t **CLOCK\_GetFlexBusClkFreq** (void)
  - Get the flexbus clock frequency.*
- uint32\_t **CLOCK\_GetFlashClkFreq** (void)
  - Get the flash clock frequency.*
- uint32\_t **CLOCK\_GetPllFllSelClkFreq** (void)
  - Get the output clock frequency selected by SIM[PLLPLLSEL].*
- uint32\_t **CLOCK\_GetEr32kClkFreq** (void)
  - Get the external reference 32K clock frequency (ERCLK32K).*
- uint32\_t **CLOCK\_GetOsc0ErClkUndivFreq** (void)
  - Get the OSC0 external reference undivided clock frequency (OSC0ERCLK\_UNDIV).*
- uint32\_t **CLOCK\_GetOsc0ErClkFreq** (void)
  - Get the OSC0 external reference clock frequency (OSC0ERCLK).*
- uint32\_t **CLOCK\_GetOsc0ErClkDivFreq** (void)
  - Get the OSC0 external reference divided clock frequency.*
- void **CLOCK\_SetSimConfig** (sim\_clock\_config\_t const \*config)
  - Set the clock configure in SIM module.*
- static void **CLOCK\_SetSimSafeDivs** (void)
  - Set the system clock dividers in SIM to safe value.*

## Variables

- volatile uint32\_t **g\_xtal0Freq**
  - External XTAL0 (OSC0) clock frequency.*
- volatile uint32\_t **g\_xtal32Freq**
  - External XTAL32/EXTAL32/RTC\_CLKIN clock frequency.*

## Driver version

- #define **FSL\_CLOCK\_DRIVER\_VERSION** (**MAKE\_VERSION**(2, 5, 2))
  - CLOCK driver version 2.5.2.*

## MCG frequency functions.

- `uint32_t CLOCK_GetOutClkFreq (void)`  
*Gets the MCG output clock (MCGOUTCLK) frequency.*
- `uint32_t CLOCK_GetFllFreq (void)`  
*Gets the MCG FLL clock (MCGFLLCLK) frequency.*
- `uint32_t CLOCK_GetInternalRefClkFreq (void)`  
*Gets the MCG internal reference clock (MCGIRCLK) frequency.*
- `uint32_t CLOCK_GetFixedFreqClkFreq (void)`  
*Gets the MCG fixed frequency clock (MCGFFCLK) frequency.*
- `uint32_t CLOCK_GetPll0Freq (void)`  
*Gets the MCG PLL0 clock (MCGPLL0CLK) frequency.*

## MCG clock configuration.

- `static void CLOCK_SetLowPowerEnable (bool enable)`  
*Enables or disables the MCG low power.*
- `status_t CLOCK_SetInternalRefClkConfig (uint8_t enableMode, mcg_irc_mode_t ircs, uint8_t fcdiv)`  
*Configures the Internal Reference clock (MCGIRCLK).*
- `status_t CLOCK_SetExternalRefClkConfig (mcg_oscsel_t oscsel)`  
*Selects the MCG external reference clock.*
- `static void CLOCK_SetFllExtRefDiv (uint8_t frdiv)`  
*Set the FLL external reference clock divider value.*
- `void CLOCK_EnablePll0 (mcg_pll_config_t const *config)`  
*Enables the PLL0 in FLL mode.*
- `static void CLOCK_DisablePll0 (void)`  
*Disables the PLL0 in FLL mode.*
- `uint32_t CLOCK_CalcPllDiv (uint32_t refFreq, uint32_t desireFreq, uint8_t *prdiv, uint8_t *vdiv)`  
*Calculates the PLL divider setting for a desired output frequency.*

## MCG clock lock monitor functions.

- `void CLOCK_SetOsc0MonitorMode (mcg_monitor_mode_t mode)`  
*Sets the OSC0 clock monitor mode.*
- `void CLOCK_SetRtcOscMonitorMode (mcg_monitor_mode_t mode)`  
*Sets the RTC OSC clock monitor mode.*
- `void CLOCK_SetPll0MonitorMode (mcg_monitor_mode_t mode)`  
*Sets the PLL0 clock monitor mode.*
- `uint32_t CLOCK_GetStatusFlags (void)`  
*Gets the MCG status flags.*
- `void CLOCK_ClearStatusFlags (uint32_t mask)`  
*Clears the MCG status flags.*

## OSC configuration

- `static void OSC_SetExtRefClkConfig (OSC_Type *base, oscer_config_t const *config)`  
*Configures the OSC external reference clock (OSCERCLK).*
- `static void OSC_SetCapLoad (OSC_Type *base, uint8_t capLoad)`  
*Sets the capacitor load configuration for the oscillator.*
- `void CLOCK_InitOsc0 (osc_config_t const *config)`

- `void CLOCK_DeinitOsc0 (void)`  
*Deinitializes the OSC0.*

## External clock frequency

- `static void CLOCK_SetXtal0Freq (uint32_t freq)`  
*Sets the XTAL0 frequency based on board settings.*
- `static void CLOCK_SetXtal32Freq (uint32_t freq)`  
*Sets the XTAL32/RTC\_CLKIN frequency based on board settings.*

## IRCs frequency

- `void CLOCK_SetSlowIrcFreq (uint32_t freq)`  
*Set the Slow IRC frequency based on the trimmed value.*
- `void CLOCK_SetFastIrcFreq (uint32_t freq)`  
*Set the Fast IRC frequency based on the trimmed value.*

## MCG auto-trim machine.

- `status_t CLOCK_TrimInternalRefClk (uint32_t extFreq, uint32_t desireFreq, uint32_t *actualFreq, mcg_atm_select_t atms)`  
*Auto trims the internal reference clock.*

## MCG mode functions.

- `mcg_mode_t CLOCK_GetMode (void)`  
*Gets the current MCG mode.*
- `status_t CLOCK_SetFeiMode (mcg_dmx32_t dmx32, mcg_drs_t drs, void(*fllStableDelay)(void))`  
*Sets the MCG to FEI mode.*
- `status_t CLOCK_SetFeeMode (uint8_t frdiv, mcg_dmx32_t dmx32, mcg_drs_t drs, void(*fllStableDelay)(void))`  
*Sets the MCG to FEE mode.*
- `status_t CLOCK_SetFbiMode (mcg_dmx32_t dmx32, mcg_drs_t drs, void(*fllStableDelay)(void))`  
*Sets the MCG to FBI mode.*
- `status_t CLOCK_SetFbeMode (uint8_t frdiv, mcg_dmx32_t dmx32, mcg_drs_t drs, void(*fllStableDelay)(void))`  
*Sets the MCG to FBE mode.*
- `status_t CLOCK_SetBlpiMode (void)`  
*Sets the MCG to BLPI mode.*
- `status_t CLOCK_SetBlpeMode (void)`  
*Sets the MCG to BLPE mode.*
- `status_t CLOCK_SetPbeMode (mcg_pll_clk_select_t pllcs, mcg_pll_config_t const *config)`  
*Sets the MCG to PBE mode.*
- `status_t CLOCK_SetPeeMode (void)`  
*Sets the MCG to PEE mode.*
- `status_t CLOCK_ExternalModeToFbeModeQuick (void)`  
*Switches the MCG to FBE mode from the external mode.*
- `status_t CLOCK_InternalModeToFbiModeQuick (void)`  
*Switches the MCG to FBI mode from internal modes.*

- `status_t CLOCK_BootToFeiMode (mcg_dmx32_t dmx32, mcg_drs_t drs, void(*fllStableDelay)(void))`  
*Sets the MCG to FEI mode during system boot up.*
- `status_t CLOCK_BootToFeeMode (mcg_oscsel_t oscsel, uint8_t frdiv, mcg_dmx32_t dmx32, mcg_drs_t drs, void(*fllStableDelay)(void))`  
*Sets the MCG to FEE mode during system bootup.*
- `status_t CLOCK_BootToBlpiMode (uint8_t fcrdiv, mcg_irc_mode_t ircs, uint8_t ircEnableMode)`  
*Sets the MCG to BLPI mode during system boot up.*
- `status_t CLOCK_BootToBlpeMode (mcg_oscsel_t oscsel)`  
*Sets the MCG to BLPE mode during system boot up.*
- `status_t CLOCK_BootToPeeMode (mcg_oscsel_t oscsel, mcg_pll_clk_select_t pllcs, mcg_pll_config_t const *config)`  
*Sets the MCG to PEE mode during system boot up.*
- `status_t CLOCK_SetMcgConfig (mcg_config_t const *config)`  
*Sets the MCG to a target mode.*

## 4.2 Data Structure Documentation

### 4.2.1 struct sim\_clock\_config\_t

#### Data Fields

- `uint8_t pllfllSel`  
*PLL/FLL/IRC48M selection.*
- `uint8_t er32kSrc`  
*ERCLK32K source selection.*
- `uint32_t clkdiv1`  
*SIM\_CLKDIV1.*

#### Field Documentation

- (1) `uint8_t sim_clock_config_t::pllfllSel`
- (2) `uint8_t sim_clock_config_t::er32kSrc`
- (3) `uint32_t sim_clock_config_t::clkdiv1`

### 4.2.2 struct oscer\_config\_t

#### Data Fields

- `uint8_t enableMode`  
*OSCERCLK enable mode.*
- `uint8_t erclkDiv`  
*Divider for OSCERCLK.*

#### Field Documentation

(1) `uint8_t oscer_config_t::enableMode`

OR'ed value of `_oscer_enable_mode`.

(2) `uint8_t oscer_config_t::erclkDiv`

#### 4.2.3 struct osc\_config\_t

Defines the configuration data structure to initialize the OSC. When porting to a new board, set the following members according to the board setting:

1. freq: The external frequency.
2. workMode: The OSC module mode.

#### Data Fields

- `uint32_t freq`  
*External clock frequency.*
- `uint8_t capLoad`  
*Capacitor load setting.*
- `osc_mode_t workMode`  
*OSC work mode setting.*
- `oscer_config_t oscerConfig`  
*Configuration for OSCERCLK.*

#### Field Documentation

(1) `uint32_t osc_config_t::freq`

(2) `uint8_t osc_config_t::capLoad`

(3) `osc_mode_t osc_config_t::workMode`

(4) `oscer_config_t osc_config_t::oscerConfig`

#### 4.2.4 struct mcg\_pll\_config\_t

#### Data Fields

- `uint8_t enableMode`  
*Enable mode.*
- `uint8_t prdiv`  
*Reference divider PRDIV.*
- `uint8_t vdiv`  
*VCO divider VDIV.*

#### Field Documentation

(1) **uint8\_t mcg\_pll\_config\_t::enableMode**

OR'ed value of enumeration \_mcg\_pll\_enable\_mode.

(2) **uint8\_t mcg\_pll\_config\_t::prdiv**(3) **uint8\_t mcg\_pll\_config\_t::vdiv****4.2.5 struct mcg\_config\_t**

When porting to a new board, set the following members according to the board setting:

1. frdiv: If the FLL uses the external reference clock, set this value to ensure that the external reference clock divided by frdiv is in the 31.25 kHz to 39.0625 kHz range.
2. The PLL reference clock divider PRDIV: PLL reference clock frequency after PRDIV should be in the FSL\_FEATURE\_MCG\_PLL\_REF\_MIN to FSL\_FEATURE\_MCG\_PLL\_REF\_MAX range.

**Data Fields**

- **mcg\_mode\_t mcgMode**  
*MCG mode.*
- **uint8\_t irclkEnableMode**  
*MCGIRCLK enable mode.*
- **mcg\_irc\_mode\_t ircs**  
*Source, MCG\_C2[IRCS].*
- **uint8\_t fcrdiv**  
*Divider, MCG\_SC[FCRDIV].*
- **uint8\_t frdiv**  
*Divider MCG\_C1[FRDIV].*
- **mcg\_drs\_t drs**  
*DCO range MCG\_C4[DRST\_DRS].*
- **mcg\_dmx32\_t dmx32**  
*MCG\_C4[DMX32].*
- **mcg\_oscsel\_t oscsel**  
*OSC select MCG\_C7[OSCSEL].*
- **mcg\_pll\_config\_t pll0Config**  
*MCGPLL0CLK configuration.*

**Field Documentation**(1) **mcg\_mode\_t mcg\_config\_t::mcgMode**(2) **uint8\_t mcg\_config\_t::irclkEnableMode**(3) **mcg\_irc\_mode\_t mcg\_config\_t::ircs**(4) **uint8\_t mcg\_config\_t::fcrdiv**(5) **uint8\_t mcg\_config\_t::frdiv**

- (6) `mcg_drs_t mcg_config_t::drs`
- (7) `mcg_dmx32_t mcg_config_t::dmx32`
- (8) `mcg_oscsel_t mcg_config_t::oscsel`
- (9) `mcg_pll_config_t mcg_config_t::pll0Config`

## 4.3 Macro Definition Documentation

### 4.3.1 #define MCG\_CONFIG\_CHECK\_PARAM 0U

Some MCG settings must be changed with conditions, for example:

1. MCGIRCLK settings, such as the source, divider, and the trim value should not change when MCGIRCLK is used as a system clock source.
2. MCG\_C7[OSCSEL] should not be changed when the external reference clock is used as a system clock source. For example, in FBE/BLPE/PBE modes.
3. The users should only switch between the supported clock modes.

MCG functions check the parameter and MCG status before setting, if not allowed to change, the functions return error. The parameter checking increases code size, if code size is a critical requirement, change [MCG\\_CONFIG\\_CHECK\\_PARAM](#) to 0 to disable parameter checking.

### 4.3.2 #define FSL\_SDK\_DISABLE\_DRIVER\_CLOCK\_CONTROL 0

When set to 0, peripheral drivers will enable clock in initialize function and disable clock in de-initialize function. When set to 1, peripheral driver will not control the clock, application could control the clock out of the driver.

Note

All drivers share this feature switcher. If it is set to 1, application should handle clock enable and disable for all drivers.

### 4.3.3 #define FSL\_CLOCK\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 2))

### 4.3.4 #define MCG\_INTERNAL\_IRC\_48M 48000000U

### 4.3.5 #define DMAMUX\_CLOCKS

**Value:**

```
{
 \kCLOCK_Dmamux0 \
}
```

#### 4.3.6 #define RTC\_CLOCKS

**Value:**

```
{ \
 kCLOCK_Rtc0 \
}
```

#### 4.3.7 #define SAI\_CLOCKS

**Value:**

```
{ \
 kCLOCK_Sai0 \
}
```

#### 4.3.8 #define PORT\_CLOCKS

**Value:**

```
{ \
 kCLOCK_PortA, kCLOCK_PortB, kCLOCK_PortC, kCLOCK_PortD, kCLOCK_PortE \
}
```

#### 4.3.9 #define FLEXBUS\_CLOCKS

**Value:**

```
{ \
 kCLOCK_Flexbus0 \
}
```

#### 4.3.10 #define EWM\_CLOCKS

**Value:**

```
{ \
 kCLOCK_Ewm0 \
}
```

#### 4.3.11 #define PIT\_CLOCKS

**Value:**

```
{
 kCLOCK_Pit0 \
}
```

#### 4.3.12 #define DSPI\_CLOCKS

**Value:**

```
{
 kCLOCK_Spi0, kCLOCK_Spi1 \
}
```

#### 4.3.13 #define LPTMR\_CLOCKS

**Value:**

```
{
 kCLOCK_Lptmr0 \
}
```

#### 4.3.14 #define FTM\_CLOCKS

**Value:**

```
{
 kCLOCK_Ftm0, kCLOCK_Ftm1, kCLOCK_Ftm2, kCLOCK_Ftm3 \
}
```

#### 4.3.15 #define EDMA\_CLOCKS

**Value:**

```
{
 kCLOCK_Dma0 \
}
```

#### 4.3.16 #define LPUART\_CLOCKS

**Value:**

```
{ \
 kCLOCK_Lpuart0 \
}
```

#### 4.3.17 #define DAC\_CLOCKS

**Value:**

```
{ \
 kCLOCK_Dac0, kCLOCK_Dac1 \
}
```

#### 4.3.18 #define ADC16\_CLOCKS

**Value:**

```
{ \
 kCLOCK_Adc0, kCLOCK_Adc1 \
}
```

#### 4.3.19 #define VREF\_CLOCKS

**Value:**

```
{ \
 kCLOCK_Vref0 \
}
```

#### 4.3.20 #define UART\_CLOCKS

**Value:**

```
{ \
 kCLOCK_Uart0, kCLOCK_Uart1, kCLOCK_Uart2 \
}
```

#### 4.3.21 #define RNGA\_CLOCKS

**Value:**

```
{ \
 kCLOCK_Rnga0 \
}
```

#### 4.3.22 #define CRC\_CLOCKS

**Value:**

```
{ \
 kCLOCK_Crc0 \
}
```

#### 4.3.23 #define I2C\_CLOCKS

**Value:**

```
{ \
 kCLOCK_I2c0, kCLOCK_I2c1 \
}
```

#### 4.3.24 #define FTF\_CLOCKS

**Value:**

```
{ \
 kCLOCK_Ftf0 \
}
```

#### 4.3.25 #define PDB\_CLOCKS

**Value:**

```
{ \
 kCLOCK_Pdb0 \
}
```

#### 4.3.26 #define CMP\_CLOCKS

**Value:**

```
{
 kCLOCK_Cmp0, kCLOCK_Cmp1 \
}
```

#### 4.3.27 #define SYS\_CLK kCLOCK\_CoreSysClk

### 4.4 Enumeration Type Documentation

#### 4.4.1 enum clock\_name\_t

Enumerator

*kCLOCK\_CoreSysClk* Core/system clock.  
*kCLOCK\_PlatClk* Platform clock.  
*kCLOCK\_BusClk* Bus clock.  
*kCLOCK\_FlexBusClk* FlexBus clock.  
*kCLOCK\_FlashClk* Flash clock.  
*kCLOCK\_FastPeriphClk* Fast peripheral clock.  
*kCLOCK\_PllFllSelClk* The clock after SIM[PLLFLSEL].  
*kCLOCK\_Er32kClk* External reference 32K clock (ERCLK32K)  
*kCLOCK\_Osc0ErClk* OSC0 external reference clock (OSC0ERCLK)  
*kCLOCK\_Osc1ErClk* OSC1 external reference clock (OSC1ERCLK)  
*kCLOCK\_Osc0ErClkUndiv* OSC0 external reference undivided clock(OSC0ERCLK\_UNDIV).  
*kCLOCK\_McgFixedFreqClk* MCG fixed frequency clock (MCGFFCLK)  
*kCLOCK\_McgInternalRefClk* MCG internal reference clock (MCGIRCLK)  
*kCLOCK\_McgFllClk* MCGFLLCLK.  
*kCLOCK\_McgPll0Clk* MCGPLL0CLK.  
*kCLOCK\_McgPll1Clk* MCGPLL1CLK.  
*kCLOCK\_McgExtPllClk* EXT\_PLLCLK.  
*kCLOCK\_McgPeriphClk* MCG peripheral clock (MCGPCLK)  
*kCLOCK\_McgIrc48MClk* MCG IRC48M clock.  
*kCLOCK\_LpoClk* LPO clock.

#### 4.4.2 enum clock\_usb\_src\_t

Enumerator

*kCLOCK\_UsbSrcPll0* Use PLL0.  
*kCLOCK\_UsbSrcIrc48M* Use IRC48M.  
*kCLOCK\_UsbSrcExt* Use USB\_CLKIN.

**4.4.3 enum clock\_ip\_name\_t****4.4.4 enum osc\_mode\_t**

Enumerator

*kOSC\_ModeExt* Use an external clock.*kOSC\_ModeOscLowPower* Oscillator low power.*kOSC\_ModeOscHighGain* Oscillator high gain.**4.4.5 enum \_osc\_cap\_load**

Enumerator

*kOSC\_Cap2P* 2 pF capacitor load*kOSC\_Cap4P* 4 pF capacitor load*kOSC\_Cap8P* 8 pF capacitor load*kOSC\_Cap16P* 16 pF capacitor load**4.4.6 enum \_oscer\_enable\_mode**

Enumerator

*kOSC\_ErClkEnable* Enable.*kOSC\_ErClkEnableInStop* Enable in stop mode.**4.4.7 enum mcg\_fll\_src\_t**

Enumerator

*kMCG\_FllSrcExternal* External reference clock is selected.*kMCG\_FllSrcInternal* The slow internal reference clock is selected.**4.4.8 enum mcg\_irc\_mode\_t**

Enumerator

*kMCG\_IrcSlow* Slow internal reference clock selected.*kMCG\_IrcFast* Fast internal reference clock selected.

**4.4.9 enum mcg\_dmx32\_t**

Enumerator

*kMCG\_Dmx32Default* DCO has a default range of 25%.*kMCG\_Dmx32Fine* DCO is fine-tuned for maximum frequency with 32.768 kHz reference.**4.4.10 enum mcg\_drs\_t**

Enumerator

*kMCG\_DrsLow* Low frequency range.*kMCG\_DrsMid* Mid frequency range.*kMCG\_DrsMidHigh* Mid-High frequency range.*kMCG\_DrsHigh* High frequency range.**4.4.11 enum mcg\_pll\_ref\_src\_t**

Enumerator

*kMCG\_PllRefOsc0* Selects OSC0 as PLL reference clock.*kMCG\_PllRefOsc1* Selects OSC1 as PLL reference clock.**4.4.12 enum mcg\_clkout\_src\_t**

Enumerator

*kMCG\_ClkOutSrcOut* Output of the FLL is selected (reset default)*kMCG\_ClkOutSrcInternal* Internal reference clock is selected.*kMCG\_ClkOutSrcExternal* External reference clock is selected.**4.4.13 enum mcg\_atm\_select\_t**

Enumerator

*kMCG\_AtmSel32k* 32 kHz Internal Reference Clock selected*kMCG\_AtmSel4m* 4 MHz Internal Reference Clock selected

#### 4.4.14 enum mcg\_oscsel\_t

Enumerator

*kMCG\_OscselOsc* Selects System Oscillator (OSCCLK)

*kMCG\_OscselRtc* Selects 32 kHz RTC Oscillator.

*kMCG\_OscselIrc* Selects 48 MHz IRC Oscillator.

#### 4.4.15 enum mcg\_pll\_clk\_select\_t

Enumerator

*kMCG\_PlClkSelPl0* PLL0 output clock is selected.

#### 4.4.16 enum mcg\_monitor\_mode\_t

Enumerator

*kMCG\_MonitorNone* Clock monitor is disabled.

*kMCG\_MonitorInt* Trigger interrupt when clock lost.

*kMCG\_MonitorReset* System reset when clock lost.

#### 4.4.17 anonymous enum

Enumeration \_mcg\_status

Enumerator

*kStatus\_MCG\_ModeUnreachable* Can't switch to target mode.

*kStatus\_MCG\_ModeInvalid* Current mode invalid for the specific function.

*kStatus\_MCG\_AtmBusClockInvalid* Invalid bus clock for ATM.

*kStatus\_MCG\_AtmDesiredFreqInvalid* Invalid desired frequency for ATM.

*kStatus\_MCG\_AtmIrcUsed* IRC is used when using ATM.

*kStatus\_MCG\_AtmHardwareFail* Hardware fail occurs during ATM.

*kStatus\_MCG\_SourceUsed* Can't change the clock source because it is in use.

#### 4.4.18 anonymous enum

Enumeration \_mcg\_status\_flags\_t

Enumerator

*kMCG\_Osc0LostFlag* OSC0 lost.

*kMCG\_Osc0InitFlag* OSC0 crystal initialized.

*kMCG\_RtcOscLostFlag* RTC OSC lost.

*kMCG\_Pl0LostFlag* PLL0 lost.

*kMCG\_Pl0LockFlag* PLL0 locked.

#### 4.4.19 anonymous enum

Enumeration \_mcg\_irclk\_enable\_mode

Enumerator

*kMCG\_IrclkEnable* MCGIRCLK enable.

*kMCG\_IrclkEnableInStop* MCGIRCLK enable in stop mode.

#### 4.4.20 anonymous enum

Enumeration \_mcg\_pll\_enable\_mode

Enumerator

*kMCG\_PlEnableIndependent* MCGPLLCLK enable independent of the MCG clock mode.

Generally, the PLL is disabled in FLL modes (FEI/FBI/FEE/FBE). Setting the PLL clock enable independent, enables the PLL in the FLL modes.

*kMCG\_PlEnableInStop* MCGPLLCLK enable in STOP mode.

#### 4.4.21 enum mcg\_mode\_t

Enumerator

*kMCG\_ModeFEI* FEI - FLL Engaged Internal.

*kMCG\_ModeFBI* FBI - FLL Bypassed Internal.

*kMCG\_ModeBLPI* BLPI - Bypassed Low Power Internal.

*kMCG\_ModeFEE* FEE - FLL Engaged External.

*kMCG\_ModeFBE* FBE - FLL Bypassed External.

*kMCG\_ModeBLPE* BLPE - Bypassed Low Power External.

*kMCG\_ModePBE* PBE - PLL Bypassed External.

*kMCG\_ModePEE* PEE - PLL Engaged External.

*kMCG\_ModeError* Unknown mode.

### 4.5 Function Documentation

#### 4.5.1 static void CLOCK\_EnableClock ( *clock\_ip\_name\_t name* ) [inline], [static]

Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>name</i> | Which clock to enable, see <a href="#">clock_ip_name_t</a> . |
|-------------|--------------------------------------------------------------|

#### 4.5.2 static void CLOCK\_DisableClock( [clock\\_ip\\_name\\_t name](#) ) [inline], [static]

Parameters

|             |                                                               |
|-------------|---------------------------------------------------------------|
| <i>name</i> | Which clock to disable, see <a href="#">clock_ip_name_t</a> . |
|-------------|---------------------------------------------------------------|

#### 4.5.3 static void CLOCK\_SetLpuartClock( [uint32\\_t src](#) ) [inline], [static]

Parameters

|            |                                       |
|------------|---------------------------------------|
| <i>src</i> | The value to set LPUART clock source. |
|------------|---------------------------------------|

#### 4.5.4 static void CLOCK\_SetEr32kClock( [uint32\\_t src](#) ) [inline], [static]

Parameters

|            |                                         |
|------------|-----------------------------------------|
| <i>src</i> | The value to set ERCLK32K clock source. |
|------------|-----------------------------------------|

#### 4.5.5 static void CLOCK\_SetTraceClock( [uint32\\_t src](#) ) [inline], [static]

Parameters

|            |                                            |
|------------|--------------------------------------------|
| <i>src</i> | The value to set debug trace clock source. |
|------------|--------------------------------------------|

#### 4.5.6 static void CLOCK\_SetPIIFISelClock( [uint32\\_t src](#) ) [inline], [static]

Parameters

|            |                                          |
|------------|------------------------------------------|
| <i>src</i> | The value to set PLLFLLSEL clock source. |
|------------|------------------------------------------|

#### 4.5.7 static void CLOCK\_SetClkOutClock( uint32\_t *src* ) [inline], [static]

Parameters

|            |                                 |
|------------|---------------------------------|
| <i>src</i> | The value to set CLKOUT source. |
|------------|---------------------------------|

#### 4.5.8 static void CLOCK\_SetRtcClkOutClock( uint32\_t *src* ) [inline], [static]

Parameters

|            |                                     |
|------------|-------------------------------------|
| <i>src</i> | The value to set RTC_CLKOUT source. |
|------------|-------------------------------------|

#### 4.5.9 bool CLOCK\_EnableUsbfs0Clock( clock\_usb\_src\_t *src*, uint32\_t *freq* )

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>src</i>  | USB FS clock source.            |
| <i>freq</i> | The frequency specified by src. |

Return values

|              |                                                         |
|--------------|---------------------------------------------------------|
| <i>true</i>  | The clock is set successfully.                          |
| <i>false</i> | The clock source is invalid to get proper USB FS clock. |

#### 4.5.10 static void CLOCK\_DisableUsbfs0Clock( void ) [inline], [static]

Disable USB FS clock.

**4.5.11 static void CLOCK\_SetOutDiv( uint32\_t *outdiv1*, uint32\_t *outdiv2*, uint32\_t *outdiv3*, uint32\_t *outdiv4* ) [inline], [static]**

Set the SIM\_CLKDIV1[OUTDIV1], SIM\_CLKDIV1[OUTDIV2], SIM\_CLKDIV1[OUTDIV3], SIM\_CLKDIV1[OUTDIV4].

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>outdiv1</i> | Clock 1 output divider value. |
| <i>outdiv2</i> | Clock 2 output divider value. |
| <i>outdiv3</i> | Clock 3 output divider value. |
| <i>outdiv4</i> | Clock 4 output divider value. |

#### 4.5.12 **uint32\_t CLOCK\_GetFreq( clock\_name\_t *clockName* )**

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in `clock_name_t`. The MCG must be properly configured before using this function.

Parameters

|                  |                                                  |
|------------------|--------------------------------------------------|
| <i>clockName</i> | Clock names defined in <code>clock_name_t</code> |
|------------------|--------------------------------------------------|

Returns

Clock frequency value in Hertz

#### 4.5.13 **uint32\_t CLOCK\_GetCoreSysClkFreq( void )**

Returns

Clock frequency in Hz.

#### 4.5.14 **uint32\_t CLOCK\_GetPlatClkFreq( void )**

Returns

Clock frequency in Hz.

#### 4.5.15 **uint32\_t CLOCK\_GetBusClkFreq( void )**

Returns

Clock frequency in Hz.

**4.5.16 uint32\_t CLOCK\_GetFlexBusClkFreq ( void )**

Returns

Clock frequency in Hz.

**4.5.17 uint32\_t CLOCK\_GetFlashClkFreq ( void )**

Returns

Clock frequency in Hz.

**4.5.18 uint32\_t CLOCK\_GetPIIFIIISelClkFreq ( void )**

Returns

Clock frequency in Hz.

**4.5.19 uint32\_t CLOCK\_GetEr32kClkFreq ( void )**

Returns

Clock frequency in Hz.

**4.5.20 uint32\_t CLOCK\_GetOsc0ErClkUndivFreq ( void )**

Returns

Clock frequency in Hz.

**4.5.21 uint32\_t CLOCK\_GetOsc0ErClkFreq ( void )**

Returns

Clock frequency in Hz.

**4.5.22 uint32\_t CLOCK\_GetOsc0ErClkDivFreq( void )**

Returns

Clock frequency in Hz.

**4.5.23 void CLOCK\_SetSimConfig( sim\_clock\_config\_t const \* config )**

This function sets system layer clock settings in SIM module.

Parameters

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

**4.5.24 static void CLOCK\_SetSimSafeDivs( void ) [inline], [static]**

The system level clocks (core clock, bus clock, flexbus clock and flash clock) must be in allowed ranges. During MCG clock mode switch, the MCG output clock changes then the system level clocks may be out of range. This function could be used before MCG mode change, to make sure system level clocks are in allowed range.

**4.5.25 uint32\_t CLOCK\_GetOutClkFreq( void )**

This function gets the MCG output clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCGOUTCLK.

**4.5.26 uint32\_t CLOCK\_GetFllFreq( void )**

This function gets the MCG FLL clock frequency in Hz based on the current MCG register value. The FLL is enabled in FEI/FBI/FEE/FBE mode and disabled in low power state in other modes.

Returns

The frequency of MCGFLLCLK.

#### 4.5.27 `uint32_t CLOCK_GetInternalRefClkFreq ( void )`

This function gets the MCG internal reference clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCGIRCLK.

#### 4.5.28 `uint32_t CLOCK_GetFixedFreqClkFreq ( void )`

This function gets the MCG fixed frequency clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCFFCLK.

#### 4.5.29 `uint32_t CLOCK_GetPll0Freq ( void )`

This function gets the MCG PLL0 clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCGPLL0CLK.

#### 4.5.30 `static void CLOCK_SetLowPowerEnable ( bool enable ) [inline], [static]`

Enabling the MCG low power disables the PLL and FLL in bypass modes. In other words, in FBE and PBE modes, enabling low power sets the MCG to BLPE mode. In FBI and PBI modes, enabling low power sets the MCG to BLPI mode. When disabling the MCG low power, the PLL or FLL are enabled based on MCG settings.

Parameters

---

|               |                                                               |
|---------------|---------------------------------------------------------------|
| <i>enable</i> | True to enable MCG low power, false to disable MCG low power. |
|---------------|---------------------------------------------------------------|

#### 4.5.31 status\_t CLOCK\_SetInternalRefClkConfig ( uint8\_t *enableMode*, mcg\_ircl\_mode\_t *ircs*, uint8\_t *fcrdiv* )

This function sets the MCGIRCLK base on parameters. It also selects the IRC source. If the fast IRC is used, this function sets the fast IRC divider. This function also sets whether the MCGIRCLK is enabled in stop mode. Calling this function in FBI/PBI/BLPI modes may change the system clock. As a result, using the function in these modes it is not allowed.

Parameters

|                   |                                                                               |
|-------------------|-------------------------------------------------------------------------------|
| <i>enableMode</i> | MCGIRCLK enable mode, OR'ed value of the enumeration _mcg_irclk_enable_-mode. |
| <i>ircs</i>       | MCGIRCLK clock source, choose fast or slow.                                   |
| <i>fcrdiv</i>     | Fast IRC divider setting (FCRDIV).                                            |

Return values

|                                |                                                                                                                                      |
|--------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_MCG_Source-Used</i> | Because the internal reference clock is used as a clock source, the configuration should not be changed. Otherwise, a glitch occurs. |
| <i>kStatus_Success</i>         | MCGIRCLK configuration finished successfully.                                                                                        |

#### 4.5.32 status\_t CLOCK\_SetExternalRefClkConfig ( mcg\_oscsel\_t *oscsel* )

Selects the MCG external reference clock source, changes the MCG\_C7[OSCSEL], and waits for the clock source to be stable. Because the external reference clock should not be changed in FEE/FBE/BLP-E/PBE/PEE modes, do not call this function in these modes.

Parameters

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>oscsel</i> | MCG external reference clock source, MCG_C7[OSCSEL]. |
|---------------|------------------------------------------------------|

Return values

|                               |                                                                                                                                      |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_MCG_SourceUsed</i> | Because the external reference clock is used as a clock source, the configuration should not be changed. Otherwise, a glitch occurs. |
| <i>kStatus_Success</i>        | External reference clock set successfully.                                                                                           |

#### 4.5.33 static void CLOCK\_SetFllExtRefDiv( uint8\_t *frdiv* ) [inline], [static]

Sets the FLL external reference clock divider value, the register MCG\_C1[FRDIV].

Parameters

|              |                                                                |
|--------------|----------------------------------------------------------------|
| <i>frdiv</i> | The FLL external reference clock divider value, MCG_C1[FRDIV]. |
|--------------|----------------------------------------------------------------|

#### 4.5.34 void CLOCK\_EnablePll0( mcg\_pll\_config\_t const \* *config* )

This function sets us the PLL0 in FLL mode and reconfigures the PLL0. Ensure that the PLL reference clock is enabled before calling this function and that the PLL0 is not used as a clock source. The function CLOCK\_CalcPllDiv gets the correct PLL divider values.

Parameters

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

#### 4.5.35 static void CLOCK\_DisablePll0( void ) [inline], [static]

This function disables the PLL0 in FLL mode. It should be used together with the [CLOCK\\_EnablePll0](#).

#### 4.5.36 uint32\_t CLOCK\_CalcPllDiv( uint32\_t *refFreq*, uint32\_t *desireFreq*, uint8\_t \* *prdiv*, uint8\_t \* *vdiv* )

This function calculates the correct reference clock divider (PRDIV) and VCO divider (VDIV) to generate a desired PLL output frequency. It returns the closest frequency match with the corresponding PRDIV/-VDIV returned from parameters. If a desired frequency is not valid, this function returns 0.

Parameters

|                   |                                                |
|-------------------|------------------------------------------------|
| <i>refFreq</i>    | PLL reference clock frequency.                 |
| <i>desireFreq</i> | Desired PLL output frequency.                  |
| <i>prdiv</i>      | PRDIV value to generate desired PLL frequency. |
| <i>vdiv</i>       | VDIV value to generate desired PLL frequency.  |

Returns

Closest frequency match that the PLL was able generate.

#### 4.5.37 void CLOCK\_SetOsc0MonitorMode ( mcg\_monitor\_mode\_t mode )

This function sets the OSC0 clock monitor mode. See [mcg\\_monitor\\_mode\\_t](#) for details.

Parameters

|             |                      |
|-------------|----------------------|
| <i>mode</i> | Monitor mode to set. |
|-------------|----------------------|

#### 4.5.38 void CLOCK\_SetRtcOscMonitorMode ( mcg\_monitor\_mode\_t mode )

This function sets the RTC OSC clock monitor mode. See [mcg\\_monitor\\_mode\\_t](#) for details.

Parameters

|             |                      |
|-------------|----------------------|
| <i>mode</i> | Monitor mode to set. |
|-------------|----------------------|

#### 4.5.39 void CLOCK\_SetPll0MonitorMode ( mcg\_monitor\_mode\_t mode )

This function sets the PLL0 clock monitor mode. See [mcg\\_monitor\\_mode\\_t](#) for details.

Parameters

|             |                      |
|-------------|----------------------|
| <i>mode</i> | Monitor mode to set. |
|-------------|----------------------|

#### 4.5.40 uint32\_t CLOCK\_GetStatusFlags ( void )

This function gets the MCG clock status flags. All status flags are returned as a logical OR of the enumeration refer to [\\_mcg\\_status\\_flags\\_t](#). To check a specific flag, compare the return value with the flag.

Example:

```
* To check the clock lost lock status of OSC0 and PLL0.
* uint32_t mcgFlags;
*
* mcgFlags = CLOCK_GetStatusFlags();
*
* if (mcgFlags & kMCG_Osc0LostFlag)
* {
* OSC0 clock lock lost. Do something.
* }
* if (mcgFlags & kMCG_Pll0LostFlag)
* {
* PLL0 clock lock lost. Do something.
* }
*
```

Returns

Logical OR value of the enumeration `_mcg_status_flags_t`.

#### 4.5.41 void CLOCK\_ClearStatusFlags ( `uint32_t mask` )

This function clears the MCG clock lock lost status. The parameter is a logical OR value of the flags to clear. See the enumeration `_mcg_status_flags_t`.

Example:

```
* To clear the clock lost lock status flags of OSC0 and PLL0.
*
* CLOCK_ClearStatusFlags(kMCG_Osc0LostFlag | kMCG_Pll0LostFlag);
*
```

Parameters

|                   |                                                                                                                  |
|-------------------|------------------------------------------------------------------------------------------------------------------|
| <code>mask</code> | The status flags to clear. This is a logical OR of members of the enumeration <code>_mcg_status_flags_t</code> . |
|-------------------|------------------------------------------------------------------------------------------------------------------|

#### 4.5.42 static void OSC\_SetExtRefClkConfig ( `OSC_Type * base`, `oscer_config_t const * config` ) [inline], [static]

This function configures the OSC external reference clock (OSCERCLK). This is an example to enable the OSCERCLK in normal and stop modes and also set the output divider to 1:

```
oscer_config_t config =
{
 .enableMode = kOSC_ErclkEnable |
 kOSC_ErclkEnableInStop,
 .erclkDiv = 1U,
};

OSC_SetExtRefClkConfig(OSC, &config);
```

Parameters

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

#### 4.5.43 static void OSC\_SetCapLoad ( OSC\_Type \* *base*, uint8\_t *capLoad* ) [inline], [static]

This function sets the specified capacitors configuration for the oscillator. This should be done in the early system level initialization function call based on the system configuration.

Parameters

|                |                                                                                |
|----------------|--------------------------------------------------------------------------------|
| <i>base</i>    | OSC peripheral address.                                                        |
| <i>capLoad</i> | OR'ed value for the capacitor load option, see <a href="#">_osc_cap_load</a> . |

Example:

To enable only 2 pF and 8 pF capacitor load, please use like this.  
`OSC_SetCapLoad(OSC, kOSC_Cap2P | kOSC_Cap8P);`

#### 4.5.44 void CLOCK\_InitOsc0 ( osc\_config\_t const \* *config* )

This function initializes the OSC0 according to the board configuration.

Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>config</i> | Pointer to the OSC0 configuration structure. |
|---------------|----------------------------------------------|

#### 4.5.45 void CLOCK\_DeinitOsc0 ( void )

This function deinitializes the OSC0.

#### 4.5.46 static void CLOCK\_SetXtal0Freq ( uint32\_t *freq* ) [inline], [static]

Parameters

|             |                                               |
|-------------|-----------------------------------------------|
| <i>freq</i> | The XTAL0/EXTAL0 input clock frequency in Hz. |
|-------------|-----------------------------------------------|

#### 4.5.47 static void CLOCK\_SetXtal32Freq ( `uint32_t freq` ) [inline], [static]

Parameters

|             |                                                           |
|-------------|-----------------------------------------------------------|
| <i>freq</i> | The XTAL32/EXTAL32/RTC_CLKIN input clock frequency in Hz. |
|-------------|-----------------------------------------------------------|

#### 4.5.48 void CLOCK\_SetSlowIrcFreq ( `uint32_t freq` )

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>freq</i> | The Slow IRC frequency input clock frequency in Hz. |
|-------------|-----------------------------------------------------|

#### 4.5.49 void CLOCK\_SetFastIrcFreq ( `uint32_t freq` )

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>freq</i> | The Fast IRC frequency input clock frequency in Hz. |
|-------------|-----------------------------------------------------|

#### 4.5.50 status\_t CLOCK\_TrimInternalRefClk ( `uint32_t extFreq, uint32_t desireFreq, uint32_t * actualFreq, mcg_atm_select_t atms` )

This function trims the internal reference clock by using the external clock. If successful, it returns the kStatus\_Success and the frequency after trimming is received in the parameter `actualFreq`. If an error occurs, the error code is returned.

Parameters

|                |                                                        |
|----------------|--------------------------------------------------------|
| <i>extFreq</i> | External clock frequency, which should be a bus clock. |
|----------------|--------------------------------------------------------|

|                   |                                             |
|-------------------|---------------------------------------------|
| <i>desireFreq</i> | Frequency to trim to.                       |
| <i>actualFreq</i> | Actual frequency after trimming.            |
| <i>atms</i>       | Trim fast or slow internal reference clock. |

Return values

|                                           |                                                                |
|-------------------------------------------|----------------------------------------------------------------|
| <i>kStatus_Success</i>                    | ATM success.                                                   |
| <i>kStatus_MCG_AtmBus-ClockInvalid</i>    | The bus clock is not in allowed range for the ATM.             |
| <i>kStatus_MCG_Atm-DesiredFreqInvalid</i> | MCGIRCLK could not be trimmed to the desired frequency.        |
| <i>kStatus_MCG_AtmIrc-Used</i>            | Could not trim because MCGIRCLK is used as a bus clock source. |
| <i>kStatus_MCG_Atm-HardwareFail</i>       | Hardware fails while trimming.                                 |

#### 4.5.51 **mcg\_mode\_t CLOCK\_GetMode( void )**

This function checks the MCG registers and determines the current MCG mode.

Returns

Current MCG mode or error code; See [mcg\\_mode\\_t](#).

#### 4.5.52 **status\_t CLOCK\_SetFeiMode( mcg\_dmx32\_t dmx32, mcg\_drs\_t drs, void(\*)(void) fllStableDelay )**

This function sets the MCG to FEI mode. If setting to FEI mode fails from the current mode, this function returns an error.

Parameters

|              |                          |
|--------------|--------------------------|
| <i>dmx32</i> | DMX32 in FEI mode.       |
| <i>drs</i>   | The DCO range selection. |

|                       |                                                                                       |
|-----------------------|---------------------------------------------------------------------------------------|
| <i>fllStableDelay</i> | Delay function to ensure that the FLL is stable. Passing NULL does not cause a delay. |
|-----------------------|---------------------------------------------------------------------------------------|

Return values

|                                     |                                           |
|-------------------------------------|-------------------------------------------|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

Note

If dmx32 is set to kMCG\_Dmx32Fine, the slow IRC must not be trimmed to a frequency above 32768 Hz.

#### 4.5.53 status\_t CLOCK\_SetFeeMode ( uint8\_t *frdiv*, mcg\_dmx32\_t *dmx32*, mcg\_drs\_t *drs*, void(\*)(void) *fllStableDelay* )

This function sets the MCG to FEE mode. If setting to FEE mode fails from the current mode, this function returns an error.

Parameters

|                       |                                                                                 |
|-----------------------|---------------------------------------------------------------------------------|
| <i>frdiv</i>          | FLL reference clock divider setting, FRDIV.                                     |
| <i>dmx32</i>          | DMX32 in FEE mode.                                                              |
| <i>drs</i>            | The DCO range selection.                                                        |
| <i>fllStableDelay</i> | Delay function to make sure FLL is stable. Passing NULL does not cause a delay. |

Return values

|                                     |                                           |
|-------------------------------------|-------------------------------------------|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

#### 4.5.54 status\_t CLOCK\_SetFbiMode ( mcg\_dmx32\_t *dmx32*, mcg\_drs\_t *drs*, void(\*)(void) *fllStableDelay* )

This function sets the MCG to FBI mode. If setting to FBI mode fails from the current mode, this function returns an error.

## Parameters

|                       |                                                                                                                                                 |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dmx32</i>          | DMX32 in FBI mode.                                                                                                                              |
| <i>drs</i>            | The DCO range selection.                                                                                                                        |
| <i>fllStableDelay</i> | Delay function to make sure FLL is stable. If the FLL is not used in FBI mode, this parameter can be NULL. Passing NULL does not cause a delay. |

## Return values

|                                     |                                           |
|-------------------------------------|-------------------------------------------|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

## Note

If *dmx32* is set to kMCG\_Dmx32Fine, the slow IRC must not be trimmed to frequency above 32768 Hz.

#### 4.5.55 status\_t CLOCK\_SetFbeMode ( uint8\_t *frdiv*, mcg\_dmx32\_t *dmx32*, mcg\_drs\_t *drs*, void(\*)(void) *fllStableDelay* )

This function sets the MCG to FBE mode. If setting to FBE mode fails from the current mode, this function returns an error.

## Parameters

|                       |                                                                                                                                                 |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>frdiv</i>          | FLL reference clock divider setting, FRDIV.                                                                                                     |
| <i>dmx32</i>          | DMX32 in FBE mode.                                                                                                                              |
| <i>drs</i>            | The DCO range selection.                                                                                                                        |
| <i>fllStableDelay</i> | Delay function to make sure FLL is stable. If the FLL is not used in FBE mode, this parameter can be NULL. Passing NULL does not cause a delay. |

## Return values

|                                     |                                      |
|-------------------------------------|--------------------------------------|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode. |
|-------------------------------------|--------------------------------------|

|                        |                                           |
|------------------------|-------------------------------------------|
| <i>kStatus_Success</i> | Switched to the target mode successfully. |
|------------------------|-------------------------------------------|

#### 4.5.56 status\_t CLOCK\_SetBlpiMode ( void )

This function sets the MCG to BLPI mode. If setting to BLPI mode fails from the current mode, this function returns an error.

Return values

|                                     |                                           |
|-------------------------------------|-------------------------------------------|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

#### 4.5.57 status\_t CLOCK\_SetBlpeMode ( void )

This function sets the MCG to BLPE mode. If setting to BLPE mode fails from the current mode, this function returns an error.

Return values

|                                     |                                           |
|-------------------------------------|-------------------------------------------|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

#### 4.5.58 status\_t CLOCK\_SetPbeMode ( mcg\_pll\_clk\_select\_t *pllcs*, mcg\_pll\_config\_t const \* *config* )

This function sets the MCG to PBE mode. If setting to PBE mode fails from the current mode, this function returns an error.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>pllcs</i>  | The PLL selection, PLLCS.         |
| <i>config</i> | Pointer to the PLL configuration. |

Return values

|                                     |                                           |
|-------------------------------------|-------------------------------------------|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

Note

1. The parameter `pllcs` selects the PLL. For platforms with only one PLL, the parameter `pllc`s is kept for interface compatibility.
2. The parameter `config` is the PLL configuration structure. On some platforms, it is possible to choose the external PLL directly, which renders the configuration structure not necessary. In this case, pass in NULL. For example: `CLOCK_SetPbeMode(kMCG_OscselOsc, kMCG_Pl-ClkSelExtPll, NULL);`

#### 4.5.59 status\_t CLOCK\_SetPeeMode ( void )

This function sets the MCG to PEE mode.

Return values

|                                     |                                           |
|-------------------------------------|-------------------------------------------|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

Note

This function only changes the CLKS to use the PLL/FLL output. If the PRDIV/VDIV are different than in the PBE mode, set them up in PBE mode and wait. When the clock is stable, switch to PEE mode.

#### 4.5.60 status\_t CLOCK\_ExternalModeToFbeModeQuick ( void )

This function switches the MCG from external modes (PEE/PBE/BLPE/FEE) to the FBE mode quickly. The external clock is used as the system clock source and PLL is disabled. However, the FLL settings are not configured. This is a lite function with a small code size, which is useful during the mode switch. For example, to switch from PEE mode to FEI mode:

```
* CLOCK_ExternalModeToFbeModeQuick();
* CLOCK_SetFeiMode(...);
*
```

Return values

|                                 |                                                                         |
|---------------------------------|-------------------------------------------------------------------------|
| <i>kStatus_Success</i>          | Switched successfully.                                                  |
| <i>kStatus_MCG_Mode-Invalid</i> | If the current mode is not an external mode, do not call this function. |

#### 4.5.61 status\_t CLOCK\_InternalModeToFbiModeQuick ( void )

This function switches the MCG from internal modes (PEI/PBI/BLPI/FEI) to the FBI mode quickly. The MCGIRCLK is used as the system clock source and PLL is disabled. However, FLL settings are not configured. This is a lite function with a small code size, which is useful during the mode switch. For example, to switch from PEI mode to FEE mode:

```
* CLOCK_InternalModeToFbiModeQuick();
* CLOCK_SetFeeMode(...);
*
```

Return values

|                                 |                                                                         |
|---------------------------------|-------------------------------------------------------------------------|
| <i>kStatus_Success</i>          | Switched successfully.                                                  |
| <i>kStatus_MCG_Mode-Invalid</i> | If the current mode is not an internal mode, do not call this function. |

#### 4.5.62 status\_t CLOCK\_BootToFeiMode ( mcg\_dmx32\_t dmx32, mcg\_drs\_t drs, void(\*)(void) fllStableDelay )

This function sets the MCG to FEI mode from the reset mode. It can also be used to set up MCG during system boot up.

Parameters

|                       |                                                  |
|-----------------------|--------------------------------------------------|
| <i>dmx32</i>          | DMX32 in FEI mode.                               |
| <i>drs</i>            | The DCO range selection.                         |
| <i>fllStableDelay</i> | Delay function to ensure that the FLL is stable. |

Return values

|                                     |                                           |
|-------------------------------------|-------------------------------------------|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

## Note

If dmx32 is set to kMCG\_Dmx32Fine, the slow IRC must not be trimmed to frequency above 32768 Hz.

#### 4.5.63 status\_t CLOCK\_BootToFeeMode ( *mcg\_oscsel\_t oscsel*, *uint8\_t frdiv*, *mcg\_dmx32\_t dmx32*, *mcg\_drs\_t drs*, *void(\*)(void) fllStableDelay* )

This function sets MCG to FEE mode from the reset mode. It can also be used to set up the MCG during system boot up.

## Parameters

|                       |                                                  |
|-----------------------|--------------------------------------------------|
| <i>oscsel</i>         | OSC clock select, OSCSEL.                        |
| <i>frdiv</i>          | FLL reference clock divider setting, FRDIV.      |
| <i>dmx32</i>          | DMX32 in FEE mode.                               |
| <i>drs</i>            | The DCO range selection.                         |
| <i>fllStableDelay</i> | Delay function to ensure that the FLL is stable. |

## Return values

|                                     |                                           |
|-------------------------------------|-------------------------------------------|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

#### 4.5.64 status\_t CLOCK\_BootToBlpiMode ( *uint8\_t fcrcdiv*, *mcg\_irc\_mode\_t ircs*, *uint8\_t ircEnableMode* )

This function sets the MCG to BLPI mode from the reset mode. It can also be used to set up the MCG during system boot up.

Parameters

|                      |                                                                                  |
|----------------------|----------------------------------------------------------------------------------|
| <i>fcrdiv</i>        | Fast IRC divider, FCRDIV.                                                        |
| <i>ircs</i>          | The internal reference clock to select, IRCS.                                    |
| <i>ircEnableMode</i> | The MCGIRCLK enable mode, OR'ed value of the enumeration _mcg_irclk_enable_mode. |

Return values

|                               |                                           |
|-------------------------------|-------------------------------------------|
| <i>kStatus_MCG_SourceUsed</i> | Could not change MCGIRCLK setting.        |
| <i>kStatus_Success</i>        | Switched to the target mode successfully. |

#### 4.5.65 status\_t CLOCK\_BootToBlpeMode ( mcg\_oscsel\_t *oscsel* )

This function sets the MCG to BLPE mode from the reset mode. It can also be used to set up the MCG during system boot up.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>oscsel</i> | OSC clock select, MCG_C7[OSCSEL]. |
|---------------|-----------------------------------|

Return values

|                                    |                                           |
|------------------------------------|-------------------------------------------|
| <i>kStatus_MCG_ModeUnreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>             | Switched to the target mode successfully. |

#### 4.5.66 status\_t CLOCK\_BootToPeeMode ( mcg\_oscsel\_t *oscsel*, mcg\_pll\_clk\_select\_t *pllcs*, mcg\_pll\_config\_t const \* *config* )

This function sets the MCG to PEE mode from reset mode. It can also be used to set up the MCG during system boot up.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>oscsel</i> | OSC clock select, MCG_C7[OSCSEL]. |
| <i>pllcs</i>  | The PLL selection, PLLCS.         |
| <i>config</i> | Pointer to the PLL configuration. |

Return values

|                                     |                                           |
|-------------------------------------|-------------------------------------------|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

#### 4.5.67 **status\_t CLOCK\_SetMcgConfig ( mcg\_config\_t const \* config )**

This function sets MCG to a target mode defined by the configuration structure. If switching to the target mode fails, this function chooses the correct path.

Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>config</i> | Pointer to the target MCG mode configuration structure. |
|---------------|---------------------------------------------------------|

Returns

Return *kStatus\_Success* if switched successfully; Otherwise, it returns an error code *\_mcg\_status*.

Note

If the external clock is used in the target mode, ensure that it is enabled. For example, if the OSC0 is used, set up OSC0 correctly before calling this function.

## 4.6 Variable Documentation

### 4.6.1 volatile uint32\_t g\_xtal0Freq

The XTAL0/EXTAL0 (OSC0) clock frequency in Hz. When the clock is set up, use the function *CLOCK\_SetXtal0Freq* to set the value in the clock driver. For example, if XTAL0 is 8 MHz:

```
* Set up the OSC0
* CLOCK_InitOsc0(...);
* Set the XTAL0 value to the clock driver.
* CLOCK_SetXtal0Freq(80000000);
*
```

This is important for the multicore platforms where only one core needs to set up the OSC0 using the *CLOCK\_InitOsc0*. All other cores need to call the *CLOCK\_SetXtal0Freq* to get a valid clock frequency.

#### 4.6.2 volatile uint32\_t g\_xtal32Freq

The XTAL32/EXTAL32/RTC\_CLKIN clock frequency in Hz. When the clock is set up, use the function CLOCK\_SetXtal32Freq to set the value in the clock driver.

This is important for the multicore platforms where only one core needs to set up the clock. All other cores need to call the CLOCK\_SetXtal32Freq to get a valid clock frequency.

## 4.7 Multipurpose Clock Generator (MCG)

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

### 4.7.1 Function description

MCG driver provides these functions:

- Functions to get the MCG clock frequency.
- Functions to configure the MCG clock, such as PLLCLK and MCGIRCLK.
- Functions for the MCG clock lock lost monitor.
- Functions for the OSC configuration.
- Functions for the MCG auto-trim machine.
- Functions for the MCG mode.

#### 4.7.1.1 MCG frequency functions

MCG module provides clocks, such as MCGOUTCLK, MCGIRCLK, MCGFFCLK, MCGFLLCLK, and MCGPLLCLK. The MCG driver provides functions to get the frequency of these clocks, such as [CLOCK\\_GetOutClkFreq\(\)](#), [CLOCK\\_GetInternalRefClkFreq\(\)](#), [CLOCK\\_GetFixedFreqClkFreq\(\)](#), [CLOCK\\_GetFllFreq\(\)](#), [CLOCK\\_GetPll0Freq\(\)](#), [CLOCK\\_GetPll1Freq\(\)](#), and [CLOCK\\_GetExtPllFreq\(\)](#). These functions get the clock frequency based on the current MCG registers.

#### 4.7.1.2 MCG clock configuration

The MCG driver provides functions to configure the internal reference clock (MCGIRCLK), the external reference clock, and MCGPLLCLK.

The function [CLOCK\\_SetInternalRefClkConfig\(\)](#) configures the MCGIRCLK, including the source and the driver. Do not change MCGIRCLK when the MCG mode is BLPI/FBI/PBI because the MCGIRCLK is used as a system clock in these modes and changing settings makes the system clock unstable.

The function [CLOCK\\_SetExternalRefClkConfig\(\)](#) configures the external reference clock source (MCG\_C7[OSCSEL]). Do not call this function when the MCG mode is BLPE/FBE/PBE/FEE/PEE because the external reference clock is used as a clock source in these modes. Changing the external reference clock source requires at least a 50 microseconds wait. The function [CLOCK\\_SetExternalRefClkConfig\(\)](#) implements a for loop delay internally. The for loop delay assumes that the system clock is 96 MHz, which ensures at least 50 micro seconds delay. However, when the system clock is slow, the delay time may significantly increase. This for loop count can be optimized for better performance for specific cases.

The MCGPLLCLK is disabled in FBE/FEE/FBI/FEI modes by default. Applications can enable the MCGPLLCLK in these modes using the functions [CLOCK\\_EnablePll0\(\)](#) and [CLOCK\\_EnablePll1\(\)](#). To enable the MCGPLLCLK, the PLL reference clock divider(PRDIV) and the PLL VCO divider(VDIV) must be set to a proper value. The function [CLOCK\\_CalcPllDiv\(\)](#) helps to get the PRDIV/VDIV.

#### 4.7.1.3 MCG clock lock monitor functions

The MCG module monitors the OSC and the PLL clock lock status. The MCG driver provides the functions to set the clock monitor mode, check the clock lost status, and clear the clock lost status.

#### 4.7.1.4 OSC configuration

The MCG is needed together with the OSC module to enable the OSC clock. The function [CLOCK\\_InitOsc0\(\)](#) `CLOCK_InitOsc1` uses the MCG and OSC to initialize the OSC. The OSC should be configured based on the board design.

#### 4.7.1.5 MCG auto-trim machine

The MCG provides an auto-trim machine to trim the MCG internal reference clock based on the external reference clock (BUS clock). During clock trimming, the MCG must not work in FEI/FBI/BLPI/PBI/PEI modes. The function [CLOCK\\_TrimInternalRefClk\(\)](#) is used for the auto clock trimming.

#### 4.7.1.6 MCG mode functions

The function `CLOCK_GetMcgMode` returns the current MCG mode. The MCG can only switch between the neighbouring modes. If the target mode is not current mode's neighbouring mode, the application must choose the proper switch path. For example, to switch to PEE mode from FEI mode, use FEI -> FBE -> PBE -> PEE.

For the MCG modes, the MCG driver provides three kinds of functions:

The first type of functions involve functions `CLOCK_SetXxxMode`, such as [CLOCK\\_SetFeiMode\(\)](#). These functions only set the MCG mode from neighbouring modes. If switching to the target mode directly from current mode is not possible, the functions return an error.

The second type of functions are the functions `CLOCK_BootToXxxMode`, such as [CLOCK\\_BootToFeiMode\(\)](#). These functions set the MCG to specific modes from reset mode. Because the source mode and target mode are specific, these functions choose the best switch path. The functions are also useful to set up the system clock during boot up.

The third type of functions is the [CLOCK\\_SetMcgConfig\(\)](#). This function chooses the right path to switch to the target mode. It is easy to use, but introduces a large code size.

Whenever the FLL settings change, there should be a 1 millisecond delay to ensure that the FLL is stable. The function [CLOCK\\_SetMcgConfig\(\)](#) implements a for loop delay internally to ensure that the FLL is stable. The for loop delay assumes that the system clock is 96 MHz, which ensures at least 1 millisecond delay. However, when the system clock is slow, the delay time may increase significantly. The for loop count can be optimized for better performance according to a specific use case.

## 4.7.2 Typical use case

The function `CLOCK_SetMcgConfig` is used to switch between any modes. However, this heavy-light function introduces a large code size. This section shows how to use the mode function to implement a quick and light-weight switch between typical specific modes. Note that the step to enable the external clock is not included in the following steps. Enable the corresponding clock before using it as a clock source.

### 4.7.2.1 Switch between BLPI and FEI

| Use case    | Steps                      | Functions                                                                |
|-------------|----------------------------|--------------------------------------------------------------------------|
| BLPI -> FEI | BLPI -> FBI                | <code>CLOCK_InternalModeToFbiModeQuick(...)</code>                       |
|             | FBI -> FEI                 | <code>CLOCK_SetFeiMode(...)</code>                                       |
|             | Configure MCGIRCLK if need | <code>CLOCK_SetInternalRefClkConfig(...)</code>                          |
| FEI -> BLPI | Configure MCGIRCLK if need | <code>CLOCK_SetInternalRefClkConfig(...)</code>                          |
|             | FEI -> FBI                 | <code>CLOCK_SetFbiMode(...)</code> with <code>fllStableDelay=NULL</code> |
|             | FBI -> BLPI                | <code>CLOCK_SetLowPowerEnable(true)</code>                               |

### 4.7.2.2 Switch between BLPI and FEE

| Use case    | Steps                                | Functions                                                                |
|-------------|--------------------------------------|--------------------------------------------------------------------------|
| BLPI -> FEE | BLPI -> FBI                          | <code>CLOCK_InternalModeToFbiModeQuick(...)</code>                       |
|             | Change external clock source if need | <code>CLOCK_SetExternalRefClkConfig(...)</code>                          |
|             | FBI -> FEE                           | <code>CLOCK_SetFeeMode(...)</code>                                       |
| FEE -> BLPI | Configure MCGIRCLK if need           | <code>CLOCK_SetInternalRefClkConfig(...)</code>                          |
|             | FEE -> FBI                           | <code>CLOCK_SetFbiMode(...)</code> with <code>fllStableDelay=NULL</code> |
|             | FBI -> BLPI                          | <code>CLOCK_SetLowPowerEnable(true)</code>                               |

#### 4.7.2.3 Switch between BLPI and PEE

| Use case    | Steps                                | Functions                                      |
|-------------|--------------------------------------|------------------------------------------------|
| BLPI -> PEE | BLPI -> FBI                          | CLOCK_InternalModeToFbi-ModeQuick(...)         |
|             | Change external clock source if need | CLOCK_SetExternalRefClk-Config(...)            |
|             | FBI -> FBE                           | CLOCK_SetFbeMode(...) // f1l-StableDelay=NULL  |
|             | FBE -> PBE                           | CLOCK_SetPbeMode(...)                          |
|             | PBE -> PEE                           | CLOCK_SetPeeMode(...)                          |
| PEE -> BLPI | PEE -> FBE                           | CLOCK_ExternalModeToFbe-ModeQuick(...)         |
|             | Configure MCGIRCLK if need           | CLOCK_SetInternalRefClk-Config(...)            |
|             | FBE -> FBI                           | CLOCK_SetFbiMode(...) with f1lStableDelay=NULL |
|             | FBI -> BLPI                          | CLOCK_SetLowPower-Enable(true)                 |

#### 4.7.2.4 Switch between BLPE and PEE

This table applies when using the same external clock source (MCG\_C7[OSCSEL]) in BLPE mode and PEE mode.

| Use case    | Steps       | Functions                              |
|-------------|-------------|----------------------------------------|
| BLPE -> PEE | BLPE -> PBE | CLOCK_SetPbeMode(...)                  |
|             | PBE -> PEE  | CLOCK_SetPeeMode(...)                  |
| PEE -> BLPE | PEE -> FBE  | CLOCK_ExternalModeToFbe-ModeQuick(...) |
|             | FBE -> BLPE | CLOCK_SetLowPower-Enable(true)         |

If using different external clock sources (MCG\_C7[OSCSEL]) in BLPE mode and PEE mode, call the [CLOCK\\_SetExternalRefClkConfig\(\)](#) in FBI or FEI mode to change the external reference clock.

| Use case | Steps       | Functions                              |
|----------|-------------|----------------------------------------|
|          | BLPE -> FBE | CLOCK_ExternalModeToFbe-ModeQuick(...) |

|             |               |                                               |
|-------------|---------------|-----------------------------------------------|
|             | FBE -> FBI    | CLOCK_SetFbiMode(...) with flStableDelay=NULL |
|             | Change source | CLOCK_SetExternalRefClkConfig(...)            |
|             | FBI -> FBE    | CLOCK_SetFbeMode(...) with flStableDelay=NULL |
|             | FBE -> PBE    | CLOCK_SetPbeMode(...)                         |
|             | PBE -> PEE    | CLOCK_SetPeeMode(...)                         |
| PEE -> BLPE | PEE -> FBE    | CLOCK_ExternalModeToFbeModeQuick(...)         |
|             | FBE -> FBI    | CLOCK_SetFbiMode(...) with flStableDelay=NULL |
|             | Change source | CLOCK_SetExternalRefClkConfig(...)            |
|             | PBI -> FBE    | CLOCK_SetFbeMode(...) with flStableDelay=NULL |
|             | FBE -> BLPE   | CLOCK_SetLowPowerEnable(true)                 |

#### 4.7.2.5 Switch between BLPE and FEE

This table applies when using the same external clock source (MCG\_C7[OSCSEL]) in BLPE mode and FEE mode.

| Use case    | Steps       | Functions                             |
|-------------|-------------|---------------------------------------|
| BLPE -> FEE | BLPE -> FBE | CLOCK_ExternalModeToFbeModeQuick(...) |
|             | FBE -> FEE  | CLOCK_SetFeeMode(...)                 |
| FEE -> BLPE | PEE -> FBE  | CLOCK_SetPbeMode(...)                 |
|             | FBE -> BLPE | CLOCK_SetLowPowerEnable(true)         |

If using different external clock sources (MCG\_C7[OSCSEL]) in BLPE mode and FEE mode, call the [CLOCK\\_SetExternalRefClkConfig\(\)](#) in FBI or FEI mode to change the external reference clock.

| Use case    | Steps       | Functions                             |
|-------------|-------------|---------------------------------------|
| BLPE -> FEE | BLPE -> FBE | CLOCK_ExternalModeToFbeModeQuick(...) |

|             |               |                                                |
|-------------|---------------|------------------------------------------------|
|             | FBE -> FBI    | CLOCK_SetFbiMode(...) with fllStableDelay=NULL |
|             | Change source | CLOCK_SetExternalRefClkConfig(...)             |
|             | FBI -> FEE    | CLOCK_SetFeeMode(...)                          |
| FEE -> BLPE | FEE -> FBI    | CLOCK_SetFbiMode(...) with fllStableDelay=NULL |
|             | Change source | CLOCK_SetExternalRefClkConfig(...)             |
|             | PBI -> FBE    | CLOCK_SetFbeMode(...) with fllStableDelay=NULL |
|             | FBE -> BLPE   | CLOCK_SetLowPowerEnable(true)                  |

#### 4.7.2.6 Switch between BLPI and PEI

| Use case    | Steps                      | Functions                             |
|-------------|----------------------------|---------------------------------------|
| BLPI -> PEI | BLPI -> PBI                | CLOCK_SetPbiMode(...)                 |
|             | PBI -> PEI                 | CLOCK_SetPeiMode(...)                 |
|             | Configure MCGIRCLK if need | CLOCK_SetInternalRefClkConfig(...)    |
| PEI -> BLPI | Configure MCGIRCLK if need | CLOCK_SetInternalRefClkConfig         |
|             | PEI -> FBI                 | CLOCK_InternalModeToFbiModeQuick(...) |
|             | FBI -> BLPI                | CLOCK_SetLowPowerEnable(true)         |

#### 4.7.3 Code Configuration Option

##### 4.7.3.1 MCG\_USER\_CONFIG\_FLL\_STABLE\_DELAY\_EN

When switching to use FLL with function `CLOCK_SetFeiMode()` and `CLOCK_SetFeeMode()`, there is an internal function `CLOCK_FllStableDelay()`. It is used to delay a few ms so that to wait the FLL to be stable enough. By default, it is implemented in driver code like the following:

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/mcg`. Once user is willing to create their own delay function, just assert the macro `MCG_USER_CONFIG_FLL_STABLE_DELAY_EN`, and then define function `CLOCK_FllStableDelay` in the application code.

# Chapter 5

## ADC16: 16-bit SAR Analog-to-Digital Converter Driver

### 5.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 16-bit SAR Analog-to-Digital Converter (ADC16) 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/adc16

#### 5.2.2 Interrupt Configuration

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/adc16

## Data Structures

- struct `adc16_config_t`  
*ADC16 converter configuration. [More...](#)*
- struct `adc16_hardware_compare_config_t`  
*ADC16 Hardware comparison configuration. [More...](#)*
- struct `adc16_channel_config_t`  
*ADC16 channel conversion configuration. [More...](#)*

## Enumerations

- enum `_adc16_channel_status_flags` { `kADC16_ChannelConversionDoneFlag` = ADC\_SC1\_COCAO\_MASK }  
*Channel status flags.*
- enum `_adc16_status_flags` {  
  `kADC16_ActiveFlag` = ADC\_SC2\_ADACT\_MASK,  
  `kADC16_CalibrationFailedFlag` = ADC\_SC3\_CALF\_MASK }  
*Converter status flags.*
- enum `adc16_channel_mux_mode_t` {  
  `kADC16_ChannelMuxA` = 0U,  
  `kADC16_ChannelMuxB` = 1U }  
*Channel multiplexer mode for each channel.*

- enum `adc16_clock_divider_t` {
   
  `kADC16_ClockDivider1` = 0U,
   
  `kADC16_ClockDivider2` = 1U,
   
  `kADC16_ClockDivider4` = 2U,
   
  `kADC16_ClockDivider8` = 3U }
   
    *Clock divider for the converter.*
- enum `adc16_resolution_t` {
   
  `kADC16_Resolution8or9Bit` = 0U,
   
  `kADC16_Resolution12or13Bit` = 1U,
   
  `kADC16_Resolution10or11Bit` = 2U,
   
  `kADC16_ResolutionSE8Bit` = `kADC16_Resolution8or9Bit`,
   
  `kADC16_ResolutionSE12Bit` = `kADC16_Resolution12or13Bit`,
   
  `kADC16_ResolutionSE10Bit` = `kADC16_Resolution10or11Bit`,
   
  `kADC16_ResolutionDF9Bit` = `kADC16_Resolution8or9Bit`,
   
  `kADC16_ResolutionDF13Bit` = `kADC16_Resolution12or13Bit`,
   
  `kADC16_ResolutionDF11Bit` = `kADC16_Resolution10or11Bit`,
   
  `kADC16_Resolution16Bit` = 3U,
   
  `kADC16_ResolutionSE16Bit` = `kADC16_Resolution16Bit`,
   
  `kADC16_ResolutionDF16Bit` = `kADC16_Resolution16Bit` }
   
    *Converter's resolution.*
- enum `adc16_clock_source_t` {
   
  `kADC16_ClockSourceAlt0` = 0U,
   
  `kADC16_ClockSourceAlt1` = 1U,
   
  `kADC16_ClockSourceAlt2` = 2U,
   
  `kADC16_ClockSourceAlt3` = 3U,
   
  `kADC16_ClockSourceAsynchronousClock` = `kADC16_ClockSourceAlt3` }
   
    *Clock source.*
- enum `adc16_long_sample_mode_t` {
   
  `kADC16_LongSampleCycle24` = 0U,
   
  `kADC16_LongSampleCycle16` = 1U,
   
  `kADC16_LongSampleCycle10` = 2U,
   
  `kADC16_LongSampleCycle6` = 3U,
   
  `kADC16_LongSampleDisabled` = 4U }
   
    *Long sample mode.*
- enum `adc16_reference_voltage_source_t` {
   
  `kADC16_ReferenceVoltageSourceVref` = 0U,
   
  `kADC16_ReferenceVoltageSourceValt` = 1U }
   
    *Reference voltage source.*
- enum `adc16_hardware_average_mode_t` {
   
  `kADC16_HardwareAverageCount4` = 0U,
   
  `kADC16_HardwareAverageCount8` = 1U,
   
  `kADC16_HardwareAverageCount16` = 2U,
   
  `kADC16_HardwareAverageCount32` = 3U,
   
  `kADC16_HardwareAverageDisabled` = 4U }
   
    *Hardware average mode.*
- enum `adc16_hardware_compare_mode_t` {

```

kADC16_HardwareCompareMode0 = 0U,
kADC16_HardwareCompareMode1 = 1U,
kADC16_HardwareCompareMode2 = 2U,
kADC16_HardwareCompareMode3 = 3U }

Hardware compare mode.

```

## Driver version

- #define **FSL\_ADC16\_DRIVER\_VERSION** (MAKE\_VERSION(2, 3, 0))  
*ADC16 driver version 2.3.0.*

## Initialization

- void **ADC16\_Init** (ADC\_Type \*base, const adc16\_config\_t \*config)  
*Initializes the ADC16 module.*
- void **ADC16\_Deinit** (ADC\_Type \*base)  
*De-initializes the ADC16 module.*
- void **ADC16\_GetDefaultConfig** (adc16\_config\_t \*config)  
*Gets an available pre-defined settings for the converter's configuration.*
- **status\_t ADC16\_DoAutoCalibration** (ADC\_Type \*base)  
*Automates the hardware calibration.*
- static void **ADC16\_SetOffsetValue** (ADC\_Type \*base, int16\_t value)  
*Sets the offset value for the conversion result.*

## Advanced Features

- static void **ADC16\_EnableDMA** (ADC\_Type \*base, bool enable)  
*Enables generating the DMA trigger when the conversion is complete.*
- static void **ADC16\_EnableHardwareTrigger** (ADC\_Type \*base, bool enable)  
*Enables the hardware trigger mode.*
- void **ADC16\_SetChannelMuxMode** (ADC\_Type \*base, adc16\_channel\_mux\_mode\_t mode)  
*Sets the channel mux mode.*
- void **ADC16\_SetHardwareCompareConfig** (ADC\_Type \*base, const adc16\_hardware\_compare\_config\_t \*config)  
*Configures the hardware compare mode.*
- void **ADC16\_SetHardwareAverage** (ADC\_Type \*base, adc16\_hardware\_average\_mode\_t mode)  
*Sets the hardware average mode.*
- uint32\_t **ADC16\_GetStatusFlags** (ADC\_Type \*base)  
*Gets the status flags of the converter.*
- void **ADC16\_ClearStatusFlags** (ADC\_Type \*base, uint32\_t mask)  
*Clears the status flags of the converter.*
- static void **ADC16\_EnableAsynchronousClockOutput** (ADC\_Type \*base, bool enable)  
*Enable/disable ADC Asynchronous clock output to other modules.*

## Conversion Channel

- void **ADC16\_SetChannelConfig** (ADC\_Type \*base, uint32\_t channelGroup, const adc16\_channel\_config\_t \*config)  
*Configures the conversion channel.*
- static uint32\_t **ADC16\_GetChannelConversionValue** (ADC\_Type \*base, uint32\_t channelGroup)

- `uint32_t ADC16_GetChannelStatusFlags (ADC_Type *base, uint32_t channelGroup)`  
*Gets the status flags of channel.*

## 5.3 Data Structure Documentation

### 5.3.1 struct adc16\_config\_t

#### Data Fields

- `adc16_reference_voltage_source_t referenceVoltageSource`  
*Select the reference voltage source.*
- `adc16_clock_source_t clockSource`  
*Select the input clock source to converter.*
- `bool enableAsynchronousClock`  
*Enable the asynchronous clock output.*
- `adc16_clock_divider_t clockDivider`  
*Select the divider of input clock source.*
- `adc16_resolution_t resolution`  
*Select the sample resolution mode.*
- `adc16_long_sample_mode_t longSampleMode`  
*Select the long sample mode.*
- `bool enableHighSpeed`  
*Enable the high-speed mode.*
- `bool enableLowPower`  
*Enable low power.*
- `bool enableContinuousConversion`  
*Enable continuous conversion mode.*
- `adc16_hardware_average_mode_t hardwareAverageMode`  
*Set hardware average mode.*

#### Field Documentation

- (1) `adc16_reference_voltage_source_t adc16_config_t::referenceVoltageSource`
- (2) `adc16_clock_source_t adc16_config_t::clockSource`
- (3) `bool adc16_config_t::enableAsynchronousClock`
- (4) `adc16_clock_divider_t adc16_config_t::clockDivider`
- (5) `adc16_resolution_t adc16_config_t::resolution`
- (6) `adc16_long_sample_mode_t adc16_config_t::longSampleMode`
- (7) `bool adc16_config_t::enableHighSpeed`
- (8) `bool adc16_config_t::enableLowPower`
- (9) `bool adc16_config_t::enableContinuousConversion`

(10) `adc16.hardware_average_mode_t adc16_config_t::hardwareAverageMode`

### 5.3.2 struct `adc16.hardware_compare_config_t`

#### Data Fields

- `adc16.hardware_compare_mode_t hardwareCompareMode`  
*Select the hardware compare mode.*
- `int16_t value1`  
*Setting value1 for hardware compare mode.*
- `int16_t value2`  
*Setting value2 for hardware compare mode.*

#### Field Documentation

(1) `adc16.hardware_compare_mode_t adc16.hardware_compare_config_t::hardwareCompareMode`

See "adc16.hardware\_compare\_mode\_t".

(2) `int16_t adc16.hardware_compare_config_t::value1`

(3) `int16_t adc16.hardware_compare_config_t::value2`

### 5.3.3 struct `adc16.channel_config_t`

#### Data Fields

- `uint32_t channelNumber`  
*Setting the conversion channel number.*
- `bool enableInterruptOnConversionCompleted`  
*Generate an interrupt request once the conversion is completed.*
- `bool enableDifferentialConversion`  
*Using Differential sample mode.*

#### Field Documentation

(1) `uint32_t adc16.channel_config_t::channelNumber`

The available range is 0-31. See channel connection information for each chip in Reference Manual document.

(2) `bool adc16.channel_config_t::enableInterruptOnConversionCompleted`

(3) `bool adc16.channel_config_t::enableDifferentialConversion`

## 5.4 Macro Definition Documentation

### 5.4.1 #define FSL\_ADC16\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 0))

## 5.5 Enumeration Type Documentation

### 5.5.1 enum \_adc16\_channel\_status\_flags

Enumerator

*kADC16\_ChannelConversionDoneFlag* Conversion done.

### 5.5.2 enum \_adc16\_status\_flags

Enumerator

*kADC16\_ActiveFlag* Converter is active.

*kADC16\_CalibrationFailedFlag* Calibration is failed.

### 5.5.3 enum adc16\_channel\_mux\_mode\_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.

Enumerator

*kADC16\_ChannelMuxA* For channel with channel mux a.

*kADC16\_ChannelMuxB* For channel with channel mux b.

### 5.5.4 enum adc16\_clock\_divider\_t

Enumerator

*kADC16\_ClockDivider1* For divider 1 from the input clock to the module.

*kADC16\_ClockDivider2* For divider 2 from the input clock to the module.

*kADC16\_ClockDivider4* For divider 4 from the input clock to the module.

*kADC16\_ClockDivider8* For divider 8 from the input clock to the module.

### 5.5.5 enum adc16\_resolution\_t

Enumerator

*kADC16\_Resolution8or9Bit* Single End 8-bit or Differential Sample 9-bit.

***kADC16\_Resolution12or13Bit*** Single End 12-bit or Differential Sample 13-bit.  
***kADC16\_Resolution10or11Bit*** Single End 10-bit or Differential Sample 11-bit.  
***kADC16\_ResolutionSE8Bit*** Single End 8-bit.  
***kADC16\_ResolutionSE12Bit*** Single End 12-bit.  
***kADC16\_ResolutionSE10Bit*** Single End 10-bit.  
***kADC16\_ResolutionDF9Bit*** Differential Sample 9-bit.  
***kADC16\_ResolutionDF13Bit*** Differential Sample 13-bit.  
***kADC16\_ResolutionDF11Bit*** Differential Sample 11-bit.  
***kADC16\_Resolution16Bit*** Single End 16-bit or Differential Sample 16-bit.  
***kADC16\_ResolutionSE16Bit*** Single End 16-bit.  
***kADC16\_ResolutionDF16Bit*** Differential Sample 16-bit.

### 5.5.6 enum adc16\_clock\_source\_t

Enumerator

***kADC16\_ClockSourceAlt0*** Selection 0 of the clock source.  
***kADC16\_ClockSourceAlt1*** Selection 1 of the clock source.  
***kADC16\_ClockSourceAlt2*** Selection 2 of the clock source.  
***kADC16\_ClockSourceAlt3*** Selection 3 of the clock source.  
***kADC16\_ClockSourceAsynchronousClock*** Using internal asynchronous clock.

### 5.5.7 enum adc16\_long\_sample\_mode\_t

Enumerator

***kADC16\_LongSampleCycle24*** 20 extra ADCK cycles, 24 ADCK cycles total.  
***kADC16\_LongSampleCycle16*** 12 extra ADCK cycles, 16 ADCK cycles total.  
***kADC16\_LongSampleCycle10*** 6 extra ADCK cycles, 10 ADCK cycles total.  
***kADC16\_LongSampleCycle6*** 2 extra ADCK cycles, 6 ADCK cycles total.  
***kADC16\_LongSampleDisabled*** Disable the long sample feature.

### 5.5.8 enum adc16\_reference\_voltage\_source\_t

Enumerator

***kADC16\_ReferenceVoltageSourceVref*** For external pins pair of VrefH and VrefL.  
***kADC16\_ReferenceVoltageSourceValt*** For alternate reference pair of ValtH and ValtL.

### 5.5.9 enum adc16.hardware\_average\_mode\_t

Enumerator

*kADC16\_HardwareAverageCount4* For hardware average with 4 samples.  
*kADC16\_HardwareAverageCount8* For hardware average with 8 samples.  
*kADC16\_HardwareAverageCount16* For hardware average with 16 samples.  
*kADC16\_HardwareAverageCount32* For hardware average with 32 samples.  
*kADC16\_HardwareAverageDisabled* Disable the hardware average feature.

### 5.5.10 enum adc16.hardware\_compare\_mode\_t

Enumerator

*kADC16\_HardwareCompareMode0*  $x < \text{value1}$ .  
*kADC16\_HardwareCompareMode1*  $x > \text{value1}$ .  
*kADC16\_HardwareCompareMode2* if  $\text{value1} \leq \text{value2}$ , then  $x < \text{value1} \ || \ x > \text{value2}$ ; else,  
 $\text{value1} > x > \text{value2}$ .  
*kADC16\_HardwareCompareMode3* if  $\text{value1} \leq \text{value2}$ , then  $\text{value1} \leq x \leq \text{value2}$ ; else  $x \geq \text{value1} \ || \ x \leq \text{value2}$ .

## 5.6 Function Documentation

### 5.6.1 void ADC16\_Init ( ADC\_Type \* *base*, const adc16\_config\_t \* *config* )

Parameters

|               |                                                           |
|---------------|-----------------------------------------------------------|
| <i>base</i>   | ADC16 peripheral base address.                            |
| <i>config</i> | Pointer to configuration structure. See "adc16_config_t". |

### 5.6.2 void ADC16\_Deinit ( ADC\_Type \* *base* )

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | ADC16 peripheral base address. |
|-------------|--------------------------------|

### 5.6.3 void ADC16\_GetDefaultConfig ( adc16\_config\_t \* *config* )

This function initializes the converter configuration structure with available settings. The default values are as follows.

```

* config->referenceVoltageSource = kADC16_ReferenceVoltageSourceVref
* ;
* config->clockSource = kADC16_ClockSourceAsynchronousClock
* ;
* config->enableAsynchronousClock = false;
* config->clockDivider = kADC16_ClockDivider8;
* config->resolution = kADC16_ResolutionSE12Bit;
* config->longSampleMode = kADC16_LongSampleDisabled;
* config->enableHighSpeed = false;
* config->enableLowPower = false;
* config->enableContinuousConversion = false;
*

```

## Parameters

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

**5.6.4 status\_t ADC16\_DoAutoCalibration ( ADC\_Type \* *base* )**

This auto calibration helps to adjust the plus/minus side gain automatically. Execute the calibration before using the converter. Note that the hardware trigger should be used during the calibration.

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | ADC16 peripheral base address. |
|-------------|--------------------------------|

## Returns

Execution status.

## Return values

|                        |                                   |
|------------------------|-----------------------------------|
| <i>kStatus_Success</i> | Calibration is done successfully. |
| <i>kStatus_Fail</i>    | Calibration has failed.           |

**5.6.5 static void ADC16\_SetOffsetValue ( ADC\_Type \* *base*, int16\_t *value* )  
[inline], [static]**

This offset value takes effect on the conversion result. If the offset value is not zero, the reading result is subtracted by it. Note, the hardware calibration fills the offset value automatically.

Parameters

|              |                                |
|--------------|--------------------------------|
| <i>base</i>  | ADC16 peripheral base address. |
| <i>value</i> | Setting offset value.          |

**5.6.6 static void ADC16\_EnableDMA ( ADC\_Type \* *base*, bool *enable* )  
[inline], [static]**

Parameters

|               |                                                                               |
|---------------|-------------------------------------------------------------------------------|
| <i>base</i>   | ADC16 peripheral base address.                                                |
| <i>enable</i> | Switcher of the DMA feature. "true" means enabled, "false" means not enabled. |

**5.6.7 static void ADC16\_EnableHardwareTrigger ( ADC\_Type \* *base*, bool *enable* )  
[inline], [static]**

Parameters

|               |                                                                                            |
|---------------|--------------------------------------------------------------------------------------------|
| <i>base</i>   | ADC16 peripheral base address.                                                             |
| <i>enable</i> | Switcher of the hardware trigger feature. "true" means enabled, "false" means not enabled. |

**5.6.8 void ADC16\_SetChannelMuxMode ( ADC\_Type \* *base*,  
adc16\_channel\_mux\_mode\_t *mode* )**

Some sample pins share the same channel index. The channel mux mode decides which pin is used for an indicated channel.

Parameters

|             |                                                           |
|-------------|-----------------------------------------------------------|
| <i>base</i> | ADC16 peripheral base address.                            |
| <i>mode</i> | Setting channel mux mode. See "adc16_channel_mux_mode_t". |

### 5.6.9 void ADC16\_SetHardwareCompareConfig ( ADC\_Type \* *base*, const adc16.hardware.compare\_config\_t \* *config* )

The hardware compare mode provides a way to process the conversion result automatically by using hardware. Only the result in the compare range is available. To compare the range, see "adc16.hardware.compare\_mode\_t" or the appropriate reference manual for more information.

Parameters

|               |                                                                                                  |
|---------------|--------------------------------------------------------------------------------------------------|
| <i>base</i>   | ADC16 peripheral base address.                                                                   |
| <i>config</i> | Pointer to the "adc16.hardware.compare_config_t" structure. Passing "NULL" disables the feature. |

### 5.6.10 void ADC16\_SetHardwareAverage ( ADC\_Type \* *base*, adc16.hardware.average\_mode\_t *mode* )

The hardware average mode provides a way to process the conversion result automatically by using hardware. The multiple conversion results are accumulated and averaged internally making them easier to read.

Parameters

|             |                                                                         |
|-------------|-------------------------------------------------------------------------|
| <i>base</i> | ADC16 peripheral base address.                                          |
| <i>mode</i> | Setting the hardware average mode. See "adc16.hardware.average_mode_t". |

### 5.6.11 uint32\_t ADC16\_GetStatusFlags ( ADC\_Type \* *base* )

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | ADC16 peripheral base address. |
|-------------|--------------------------------|

Returns

Flags' mask if indicated flags are asserted. See "\_adc16\_status\_flags".

### 5.6.12 void ADC16\_ClearStatusFlags ( ADC\_Type \* *base*, uint32\_t *mask* )

Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | ADC16 peripheral base address.                               |
| <i>mask</i> | Mask value for the cleared flags. See "_adc16_status_flags". |

### 5.6.13 static void ADC16\_EnableAsynchronousClockOutput ( **ADC\_Type \* base**, **bool enable** ) [inline], [static]

Parameters

|               |                                                                                                                                                                                                                                                                                                                                                         |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | ADC16 peripheral base address.                                                                                                                                                                                                                                                                                                                          |
| <i>enable</i> | Used to enable/disable ADC ADACK output. <ul style="list-style-type: none"> <li>• <b>true</b> Asynchronous clock and clock output is enabled regardless of the state of the ADC.</li> <li>• <b>false</b> Asynchronous clock output disabled, asynchronous clock is enabled only if it is selected as input clock and a conversion is active.</li> </ul> |

### 5.6.14 void ADC16\_SetChannelConfig ( **ADC\_Type \* base**, **uint32\_t channelGroup**, **const adc16\_channel\_config\_t \* config** )

This operation triggers the conversion when in software trigger mode. When in hardware trigger mode, this API configures the channel while the external trigger source helps to trigger the conversion.

Note that the "Channel Group" has a detailed description. To allow sequential conversions of the ADC to be triggered by internal peripherals, the ADC has more than one group of status and control registers, one for each conversion. The channel group parameter indicates which group of registers are used, for example, channel group 0 is for Group A registers and channel group 1 is for Group B registers. The channel groups are used in a "ping-pong" approach to control the ADC operation. At any point, only one of the channel groups is actively controlling ADC conversions. The channel group 0 is used for both software and hardware trigger modes. Channel group 1 and greater indicates multiple channel group registers for use only in hardware trigger mode. See the chip configuration information in the appropriate MCU reference manual for the number of SC1n registers (channel groups) specific to this device. Channel group 1 or greater are not used for software trigger operation. Therefore, writing to these channel groups does not initiate a new conversion. Updating the channel group 0 while a different channel group is actively controlling a conversion is allowed and vice versa. Writing any of the channel group registers while that specific channel group is actively controlling a conversion aborts the current conversion.

Parameters

|                     |                                                                               |
|---------------------|-------------------------------------------------------------------------------|
| <i>base</i>         | ADC16 peripheral base address.                                                |
| <i>channelGroup</i> | Channel group index.                                                          |
| <i>config</i>       | Pointer to the "adc16_channel_config_t" structure for the conversion channel. |

### 5.6.15 static uint32\_t ADC16\_GetChannelConversionValue ( ADC\_Type \* *base*, uint32\_t *channelGroup* ) [inline], [static]

Parameters

|                     |                                |
|---------------------|--------------------------------|
| <i>base</i>         | ADC16 peripheral base address. |
| <i>channelGroup</i> | Channel group index.           |

Returns

Conversion value.

### 5.6.16 uint32\_t ADC16\_GetChannelStatusFlags ( ADC\_Type \* *base*, uint32\_t *channelGroup* )

Parameters

|                     |                                |
|---------------------|--------------------------------|
| <i>base</i>         | ADC16 peripheral base address. |
| <i>channelGroup</i> | Channel group index.           |

Returns

Flags' mask if indicated flags are asserted. See "\_adc16\_channel\_status\_flags".

# Chapter 6

## CMP: Analog Comparator Driver

### 6.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Analog Comparator (CMP) module of MCUXpresso SDK devices.

The CMP driver is a basic comparator with advanced features. The APIs for the basic comparator enable the CMP to compare the two voltages of the two input channels and create the output of the comparator result. The APIs for advanced features can be used as the plug-in functions based on the basic comparator. They can process the comparator's output with hardware support.

### 6.2 Typical use case

#### 6.2.1 Polling Configuration

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/cmp

#### 6.2.2 Interrupt Configuration

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/cmp

## Data Structures

- struct [cmp\\_config\\_t](#)  
*Configures the comparator. [More...](#)*
- struct [cmp\\_filter\\_config\\_t](#)  
*Configures the filter. [More...](#)*
- struct [cmp\\_dac\\_config\\_t](#)  
*Configures the internal DAC. [More...](#)*

## Enumerations

- enum [\\_cmp\\_interrupt\\_enable](#) {  
  kCMP\_OutputRisingInterruptEnable = CMP\_SCR\_IER\_MASK,  
  kCMP\_OutputFallingInterruptEnable = CMP\_SCR\_IEF\_MASK }  
*Interrupt enable/disable mask.*
- enum [\\_cmp\\_status\\_flags](#) {  
  kCMP\_OutputRisingEventFlag = CMP\_SCR\_CFR\_MASK,  
  kCMP\_OutputFallingEventFlag = CMP\_SCR\_CFF\_MASK,  
  kCMP\_OutputAssertEventFlag = CMP\_SCR\_COUT\_MASK }  
*Status flags' mask.*

- enum `cmp_hysteresis_mode_t` {
   
    `kCMP_HysteresisLevel0` = 0U,  
`kCMP_HysteresisLevel1` = 1U,  
`kCMP_HysteresisLevel2` = 2U,  
`kCMP_HysteresisLevel3` = 3U }
   
*CMP Hysteresis mode.*
- enum `cmp_reference_voltage_source_t` {
   
    `kCMP_VrefSourceVin1` = 0U,  
`kCMP_VrefSourceVin2` = 1U }
   
*CMP Voltage Reference source.*

## Driver version

- #define `FSL_CMP_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 2)`)
   
*CMP driver version 2.0.2.*

## Initialization

- void `CMP_Init` (`CMP_Type` \*base, const `cmp_config_t` \*config)
   
*Initializes the CMP.*
- void `CMP_Deinit` (`CMP_Type` \*base)
   
*De-initializes the CMP module.*
- static void `CMP_Enable` (`CMP_Type` \*base, bool enable)
   
*Enables/disables the CMP module.*
- void `CMP_GetDefaultConfig` (`cmp_config_t` \*config)
   
*Initializes the CMP user configuration structure.*
- void `CMP_SetInputChannels` (`CMP_Type` \*base, `uint8_t` positiveChannel, `uint8_t` negativeChannel)
   
*Sets the input channels for the comparator.*

## Advanced Features

- void `CMP_EnableDMA` (`CMP_Type` \*base, bool enable)
   
*Enables/disables the DMA request for rising/falling events.*
- static void `CMP_EnableWindowMode` (`CMP_Type` \*base, bool enable)
   
*Enables/disables the window mode.*
- void `CMP_SetFilterConfig` (`CMP_Type` \*base, const `cmp_filter_config_t` \*config)
   
*Configures the filter.*
- void `CMP_SetDACCConfig` (`CMP_Type` \*base, const `cmp_dac_config_t` \*config)
   
*Configures the internal DAC.*
- void `CMP_EnableInterrupts` (`CMP_Type` \*base, `uint32_t` mask)
   
*Enables the interrupts.*
- void `CMP_DisableInterrupts` (`CMP_Type` \*base, `uint32_t` mask)
   
*Disables the interrupts.*

## Results

- `uint32_t CMP_GetStatusFlags` (`CMP_Type` \*base)
   
*Gets the status flags.*
- void `CMP_ClearStatusFlags` (`CMP_Type` \*base, `uint32_t` mask)
   
*Clears the status flags.*

## 6.3 Data Structure Documentation

### 6.3.1 struct cmp\_config\_t

#### Data Fields

- bool `enableCmp`  
*Enable the CMP module.*
- `cmp_hysteresis_mode_t hysteresisMode`  
*CMP Hysteresis mode.*
- bool `enableHighSpeed`  
*Enable High-speed (HS) comparison mode.*
- bool `enableInvertOutput`  
*Enable the inverted comparator output.*
- bool `useUnfilteredOutput`  
*Set the compare output(COUT) to equal COUTA(true) or COUT(false).*
- bool `enablePinOut`  
*The comparator output is available on the associated pin.*
- bool `enableTriggerMode`  
*Enable the trigger mode.*

#### Field Documentation

- (1) `bool cmp_config_t::enableCmp`
- (2) `cmp_hysteresis_mode_t cmp_config_t::hysteresisMode`
- (3) `bool cmp_config_t::enableHighSpeed`
- (4) `bool cmp_config_t::enableInvertOutput`
- (5) `bool cmp_config_t::useUnfilteredOutput`
- (6) `bool cmp_config_t::enablePinOut`
- (7) `bool cmp_config_t::enableTriggerMode`

### 6.3.2 struct cmp\_filter\_config\_t

#### Data Fields

- bool `enableSample`  
*Using the external SAMPLE as a sampling clock input or using a divided bus clock.*
- `uint8_t filterCount`  
*Filter Sample Count.*
- `uint8_t filterPeriod`  
*Filter Sample Period.*

#### Field Documentation

(1) `bool cmp_filter_config_t::enableSample`

(2) `uint8_t cmp_filter_config_t::filterCount`

Available range is 1-7; 0 disables the filter.

(3) `uint8_t cmp_filter_config_t::filterPeriod`

The divider to the bus clock. Available range is 0-255.

### 6.3.3 struct cmp\_dac\_config\_t

#### Data Fields

- `cmp_reference_voltage_source_t referenceVoltageSource`  
*Supply voltage reference source.*
- `uint8_t DACValue`  
*Value for the DAC Output Voltage.*

#### Field Documentation

(1) `cmp_reference_voltage_source_t cmp_dac_config_t::referenceVoltageSource`

(2) `uint8_t cmp_dac_config_t::DACValue`

Available range is 0-63.

## 6.4 Macro Definition Documentation

### 6.4.1 #define FSL\_CMP\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))

## 6.5 Enumeration Type Documentation

### 6.5.1 enum \_cmp\_interrupt\_enable

Enumerator

`kCMP_OutputRisingInterruptEnable` Comparator interrupt enable rising.

`kCMP_OutputFallingInterruptEnable` Comparator interrupt enable falling.

### 6.5.2 enum \_cmp\_status\_flags

Enumerator

`kCMP_OutputRisingEventFlag` Rising-edge on the comparison output has occurred.

`kCMP_OutputFallingEventFlag` Falling-edge on the comparison output has occurred.

***kCMP\_OutputAssertEventFlag*** Return the current value of the analog comparator output.

### 6.5.3 enum cmp\_hysteresis\_mode\_t

Enumerator

***kCMP\_HysteresisLevel0*** Hysteresis level 0.

***kCMP\_HysteresisLevel1*** Hysteresis level 1.

***kCMP\_HysteresisLevel2*** Hysteresis level 2.

***kCMP\_HysteresisLevel3*** Hysteresis level 3.

### 6.5.4 enum cmp\_reference\_voltage\_source\_t

Enumerator

***kCMP\_VrefSourceVin1*** Vin1 is selected as a resistor ladder network supply reference Vin.

***kCMP\_VrefSourceVin2*** Vin2 is selected as a resistor ladder network supply reference Vin.

## 6.6 Function Documentation

### 6.6.1 void CMP\_Init ( **CMP\_Type** \* *base*, **const cmp\_config\_t** \* *config* )

This function initializes the CMP module. The operations included are as follows.

- Enabling the clock for CMP module.
- Configuring the comparator.
- Enabling the CMP module. Note that for some devices, multiple CMP instances share the same clock gate. In this case, to enable the clock for any instance enables all CMPS. See the appropriate MCU reference manual for the clock assignment of the CMP.

Parameters

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

### 6.6.2 void CMP\_Deinit ( **CMP\_Type** \* *base* )

This function de-initializes the CMP module. The operations included are as follows.

- Disabling the CMP module.
- Disabling the clock for CMP module.

This function disables the clock for the CMP. Note that for some devices, multiple CMP instances share the same clock gate. In this case, before disabling the clock for the CMP, ensure that all the CMP instances are not used.

Parameters

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

### 6.6.3 static void CMP\_Enable ( **CMP\_Type** \* *base*, **bool** *enable* ) [inline], [static]

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | CMP peripheral base address.    |
| <i>enable</i> | Enables or disables the module. |

### 6.6.4 void CMP\_GetDefaultConfig ( **cmp\_config\_t** \* *config* )

This function initializes the user configuration structure to these default values.

```
* config->enableCmp = true;
* config->hysteresisMode = kCMP_HysteresisLevel0;
* config->enableHighSpeed = false;
* config->enableInvertOutput = false;
* config->useUnfilteredOutput= false;
* config->enablePinOut = false;
* config->enableTriggerMode = false;
*
```

Parameters

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

### 6.6.5 void CMP\_SetInputChannels ( **CMP\_Type** \* *base*, **uint8\_t** *positiveChannel*, **uint8\_t** *negativeChannel* )

This function sets the input channels for the comparator. Note that two input channels cannot be set the same way in the application. When the user selects the same input from the analog mux to the positive and negative port, the comparator is disabled automatically.

Parameters

|                         |                                                             |
|-------------------------|-------------------------------------------------------------|
| <i>base</i>             | CMP peripheral base address.                                |
| <i>positive-Channel</i> | Positive side input channel number. Available range is 0-7. |
| <i>negative-Channel</i> | Negative side input channel number. Available range is 0-7. |

### 6.6.6 void CMP\_EnableDMA ( **CMP\_Type** \* *base*, **bool** *enable* )

This function enables/disables the DMA request for rising/falling events. Either event triggers the generation of the DMA request from CMP if the DMA feature is enabled. Both events are ignored for generating the DMA request from the CMP if the DMA is disabled.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | CMP peripheral base address.     |
| <i>enable</i> | Enables or disables the feature. |

### 6.6.7 static void CMP\_EnableWindowMode ( **CMP\_Type** \* *base*, **bool** *enable* ) [inline], [static]

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | CMP peripheral base address.     |
| <i>enable</i> | Enables or disables the feature. |

### 6.6.8 void CMP\_SetFilterConfig ( **CMP\_Type** \* *base*, **const cmp\_filter\_config\_t** \* *config* )

Parameters

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

### 6.6.9 void CMP\_SetDACConfig ( **CMP\_Type** \* *base*, **const cmp\_dac\_config\_t** \* *config* )

Parameters

|               |                                                                      |
|---------------|----------------------------------------------------------------------|
| <i>base</i>   | CMP peripheral base address.                                         |
| <i>config</i> | Pointer to the configuration structure. "NULL" disables the feature. |

### 6.6.10 void CMP\_EnableInterrupts ( **CMP\_Type** \* *base*, **uint32\_t** *mask* )

Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>base</i> | CMP peripheral base address.                            |
| <i>mask</i> | Mask value for interrupts. See "_cmp_interrupt_enable". |

### 6.6.11 void CMP\_DisableInterrupts ( **CMP\_Type** \* *base*, **uint32\_t** *mask* )

Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>base</i> | CMP peripheral base address.                            |
| <i>mask</i> | Mask value for interrupts. See "_cmp_interrupt_enable". |

### 6.6.12 **uint32\_t** CMP\_GetStatusFlags ( **CMP\_Type** \* *base* )

Parameters

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

Returns

Mask value for the asserted flags. See "\_cmp\_status\_flags".

### 6.6.13 void CMP\_ClearStatusFlags ( **CMP\_Type** \* *base*, **uint32\_t** *mask* )

## Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| <i>base</i> | CMP peripheral base address.                       |
| <i>mask</i> | Mask value for the flags. See "_cmp_status_flags". |

# Chapter 7

## Common Driver

### 7.1 Overview

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

### Macros

- `#define FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ 1`  
*Macro to use the default weak IRQ handler in drivers.*
- `#define MAKE_STATUS(group, code) (((group)*100L) + (code)))`  
*Construct a status code value from a group and code number.*
- `#define MAKE_VERSION(major, minor, bugfix) (((major) * 65536L) + ((minor) * 256L) + (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 DEBUG_CONSOLE_DEVICE_TYPE_QSCI 10U`  
*Debug console based on QSCI.*
- `#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 `_status_groups` {  
    `kStatusGroup_Generic` = 0,  
    `kStatusGroup_FLASH` = 1,  
    `kStatusGroup_LP SPI` = 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_SD SPI` = 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_USACARD` = 58,  
    `kStatusGroup_SDIF` = 59,

```

kStatusGroup_BMA = 164 }

Status group numbers.
• enum {
 kStatus_Success = MAKE_STATUS(kStatusGroup_Generic, 0),
 kStatus_Fail = MAKE_STATUS(kStatusGroup_Generic, 1),
 kStatus_ReadOnly = MAKE_STATUS(kStatusGroup_Generic, 2),
 kStatus_OutOfRange = MAKE_STATUS(kStatusGroup_Generic, 3),
 kStatus_InvalidArgument = MAKE_STATUS(kStatusGroup_Generic, 4),
 kStatus_Timeout = MAKE_STATUS(kStatusGroup_Generic, 5),
 kStatus_NoTransferInProgress,
 kStatus_Busy = MAKE_STATUS(kStatusGroup_Generic, 7),
 kStatus_NoData }
Generic status return codes.

```

## Functions

- 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 delayTime\_us, uint32\_t coreClock\_Hz)  
*Delay at least for some time.*

## Driver version

- #define **FSL\_COMMON\_DRIVER\_VERSION** (MAKE\_VERSION(2, 3, 2))  
*common driver version.*

## 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)

## Suppress fallthrough warning macro

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

## 7.2 Macro Definition Documentation

### 7.2.1 #define FSL\_DRIVER\_TRANSFER\_DOUBLE\_WEAK\_IRQ 1

### 7.2.2 #define MAKE\_STATUS( *group*, *code* ) (((*group*)\*100L + (*code*)))

**7.2.3 #define MAKE\_VERSION( major, minor, bugfix ) (((major) \* 65536L) + ((minor) \* 256L) + (bugfix))**

The driver version is a 32-bit number, for both 32-bit platforms(such as Cortex M) and 16-bit platforms(such as DSC).

|        |               |               |         |   |
|--------|---------------|---------------|---------|---|
| Unused | Major Version | Minor Version | Bug Fix |   |
| 31     | 25 24         | 17 16         | 9 8     | 0 |

**7.2.4 #define FSL\_COMMON\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 2))**

**7.2.5 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_NONE 0U**

**7.2.6 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_UART 1U**

**7.2.7 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_LPUART 2U**

**7.2.8 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_LPSCI 3U**

**7.2.9 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_USBCDC 4U**

**7.2.10 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_FLEXCOMM 5U**

**7.2.11 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_IUART 6U**

**7.2.12 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_VUSART 7U**

**7.2.13 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_MINI\_USART 8U**

**7.2.14 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_SWO 9U**

**7.2.15 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_QSCI 10U**

**7.2.16 #define ARRAY\_SIZE( x ) (sizeof(x) / sizeof((x)[0]))**

## 7.3 Typedef Documentation

**7.3.1 typedef int32\_t status\_t**

## 7.4 Enumeration Type Documentation

### 7.4.1 enum \_status\_groups

Enumerator

- kStatusGroup\_Generic*** Group number for generic status codes.
- kStatusGroup\_FLASH*** Group number for FLASH status codes.
- kStatusGroup\_LP SPI*** Group number for LP SPI 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 I2C 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\_SD SPI*** Group number for SD SPI status codes.
- kStatusGroup\_FLEXIO\_I2S*** Group number for FLEXIO I2S status codes.
- kStatusGroup\_FLEXIO\_MCU LCD*** 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.  
*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\_SFA* Group number for SFA status codes.  
*kStatusGroup\_SPC* Group number for SPC status codes.  
*kStatusGroup\_PUF* Group number for PUF status codes.  
*kStatusGroup\_TOUCH\_PANEL* Group number for touch panel 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\_HAL\_I2S* Group number for HAL I2S status codes.  
*kStatusGroup\_TIMERMANAGER* Group number for TiMER MANAGER status codes.  
*kStatusGroup\_SERIALMANAGER* 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.  
*kStatusGroup\_ASRC* Group number for codec status ASRC.  
*kStatusGroup\_OTFAD* Group number for codec status codes.  
*kStatusGroup\_SDIOSLV* Group number for SDIOSLV status codes.  
*kStatusGroup\_MECC* Group number for MECC status codes.  
*kStatusGroup\_ENET\_QOS* Group number for ENET\_QOS status codes.  
*kStatusGroup\_LOG* Group number for LOG status codes.  
*kStatusGroup\_I3CBUS* Group number for I3CBUS status codes.  
*kStatusGroup\_QSCI* Group number for QSCI status codes.  
*kStatusGroup\_SNT* Group number for SNT status codes.  
*kStatusGroup\_QUEUEDSPI* Group number for QSPI status codes.  
*kStatusGroup\_POWER\_MANAGER* Group number for POWER\_MANAGER status codes.  
*kStatusGroup\_IPED* Group number for IPED status codes.  
*kStatusGroup\_CSS\_PKC* Group number for CSS PKC status codes.  
*kStatusGroup\_HOSTIF* Group number for HOSTIF status codes.  
*kStatusGroup\_CLIF* Group number for CLIF status codes.  
*kStatusGroup\_BMA* Group number for BMA status codes.

## 7.4.2 anonymous enum

Enumerator

*kStatus\_Success* Generic status for Success.  
*kStatus\_Fail* Generic status for Fail.  
*kStatus\_ReadOnly* Generic status for read only failure.  
*kStatus\_OutOfRange* Generic status for out of range access.  
*kStatus\_InvalidArgument* Generic status for invalid argument check.  
*kStatus\_Timeout* Generic status for timeout.  
*kStatus\_NoTransferInProgress* Generic status for no transfer in progress.  
*kStatus\_Busy* Generic status for module is busy.

**kStatus\_NoData** Generic status for no data is found for the operation.

## 7.5 Function Documentation

### 7.5.1 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. |
|------------|-------------------|

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

Parameters

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

### 7.5.3 void SDK\_DelayAtLeastUs ( uint32\_t *delayTime\_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>delayTime_us</i> | Delay time in unit of microsecond. |
| <i>coreClock_Hz</i> | Core clock frequency with Hz.      |

# Chapter 8

## CRC: Cyclic Redundancy Check Driver

### 8.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Cyclic Redundancy Check (CRC) module of MCUXpresso SDK devices.

The cyclic redundancy check (CRC) module generates 16/32-bit CRC code for error detection. The CRC module also provides a programmable polynomial, seed, and other parameters required to implement a 16-bit or 32-bit CRC standard.

### 8.2 CRC Driver Initialization and Configuration

`CRC_Init()` function enables the clock gate for the CRC module in the SIM module and fully (re-)configures the CRC module according to the configuration structure. The seed member of the configuration structure is the initial checksum for which new data can be added to. When starting a new checksum computation, the seed is set to the initial checksum per the CRC protocol specification. For continued checksum operation, the seed is set to the intermediate checksum value as obtained from previous calls to `CRC_Get16bitResult()` or `CRC_Get32bitResult()` function. After calling the `CRC_Init()`, one or multiple `CRC_WriteData()` calls follow to update the checksum with data and `CRC_Get16bitResult()` or `CRC_Get32bitResult()` follow to read the result. The `crcResult` member of the configuration structure determines whether the `CRC_Get16bitResult()` or `CRC_Get32bitResult()` return value is a final checksum or an intermediate checksum. The `CRC_Init()` function can be called as many times as required allowing for runtime changes of the CRC protocol.

`CRC_GetDefaultConfig()` function can be used to set the module configuration structure with parameters for CRC-16/CCIT-FALSE protocol.

### 8.3 CRC Write Data

The `CRC_WriteData()` function adds data to the CRC. Internally, it tries to use 32-bit reads and writes for all aligned data in the user buffer and 8-bit reads and writes for all unaligned data in the user buffer. This function can update the CRC with user-supplied data chunks of an arbitrary size, so one can update the CRC byte by byte or with all bytes at once. Prior to calling the CRC configuration function `CRC_Init()` fully specifies the CRC module configuration for the `CRC_WriteData()` call.

### 8.4 CRC Get Checksum

The `CRC_Get16bitResult()` or `CRC_Get32bitResult()` function reads the CRC module data register. Depending on the prior CRC module usage, the return value is either an intermediate checksum or the final checksum. For example, for 16-bit CRCs the following call sequences can be used.

`CRC_Init() / CRC_WriteData() / CRC_Get16bitResult()` to get the final checksum.

`CRC_Init() / CRC_WriteData() / ... / CRC_WriteData() / CRC_Get16bitResult()` to get the final checksum.

`CRC_Init()` / `CRC_WriteData()` / `CRC_Get16bitResult()` to get an intermediate checksum.

`CRC_Init()` / `CRC_WriteData()` / ... / `CRC_WriteData()` / `CRC_Get16bitResult()` to get an intermediate checksum.

## 8.5 Comments about API usage in RTOS

If multiple RTOS tasks share the CRC module to compute checksums with different data and/or protocols, the following needs to be implemented by the user.

The triplets

`CRC_Init()` / `CRC_WriteData()` / `CRC_Get16bitResult()` or `CRC_Get32bitResult()`

The triplets are protected by the RTOS mutex to protect the CRC module against concurrent accesses from different tasks. This is an example. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/crcRefer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/crc

## Data Structures

- struct `crc_config_t`  
*CRC protocol configuration. [More...](#)*

## Macros

- #define `CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT` 1  
*Default configuration structure filled by `CRC_GetDefaultConfig()`.*

## Enumerations

- enum `crc_bits_t` {  
`kCrcBits16` = 0U,  
`kCrcBits32` = 1U }  
*CRC bit width.*
- enum `crc_result_t` {  
`kCrcFinalChecksum` = 0U,  
`kCrcIntermediateChecksum` = 1U }  
*CRC result type.*

## Functions

- void `CRC_Init` (CRC\_Type \*base, const `crc_config_t` \*config)  
*Enables and configures the CRC peripheral module.*
- static void `CRC_Deinit` (CRC\_Type \*base)  
*Disables the CRC peripheral module.*
- void `CRC_GetDefaultConfig` (`crc_config_t` \*config)

- `void CRC_WriteData(CRC_Type *base, const uint8_t *data, size_t dataSize)`  
*Writes data to the CRC module.*
- `uint32_t CRC_Get32bitResult(CRC_Type *base)`  
*Reads the 32-bit checksum from the CRC module.*
- `uint16_t CRC_Get16bitResult(CRC_Type *base)`  
*Reads a 16-bit checksum from the CRC module.*

## Driver version

- `#define FSL_CRC_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))`  
*CRC driver version.*

## 8.6 Data Structure Documentation

### 8.6.1 struct crc\_config\_t

This structure holds the configuration for the CRC protocol.

#### Data Fields

- `uint32_t polynomial`  
*CRC Polynomial, MSBit first.*
- `uint32_t seed`  
*Starting checksum value.*
- `bool reflectIn`  
*Reflect bits on input.*
- `bool reflectOut`  
*Reflect bits on output.*
- `bool complementChecksum`  
*True if the result shall be complement of the actual checksum.*
- `crc_bits_t crcBits`  
*Selects 16- or 32- bit CRC protocol.*
- `crc_result_t crcResult`  
*Selects final or intermediate checksum return from `CRC_Get16bitResult()` or `CRC_Get32bitResult()`*

#### Field Documentation

##### (1) `uint32_t crc_config_t::polynomial`

Example polynomial:  $0x1021 = 1\_0000\_0010\_0001 = x^{12}+x^5+1$

##### (2) `bool crc_config_t::reflectIn`

##### (3) `bool crc_config_t::reflectOut`

##### (4) `bool crc_config_t::complementChecksum`

##### (5) `crc_bits_t crc_config_t::crcBits`

## 8.7 Macro Definition Documentation

### 8.7.1 #define FSL\_CRC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 3))

Version 2.0.3.

Current version: 2.0.3

Change log:

- Version 2.0.3
  - Fix MISRA issues
- Version 2.0.2
  - Fix MISRA issues
- Version 2.0.1
  - move DATA and DATALL macro definition from header file to source file

### 8.7.2 #define CRC\_DRIVER\_USE\_CRC16\_CCIT\_FALSE\_AS\_DEFAULT 1

Use CRC16-CCIT-FALSE as default.

## 8.8 Enumeration Type Documentation

### 8.8.1 enum crc\_bits\_t

Enumerator

*kCrcBits16* Generate 16-bit CRC code.

*kCrcBits32* Generate 32-bit CRC code.

### 8.8.2 enum crc\_result\_t

Enumerator

*kCrcFinalChecksum* CRC data register read value is the final checksum. Reflect out and final xor protocol features are applied.

*kCrcIntermediateChecksum* CRC data register read value is intermediate checksum (raw value).

Reflect out and final xor protocol feature are not applied. Intermediate checksum can be used as a seed for [CRC\\_Init\(\)](#) to continue adding data to this checksum.

## 8.9 Function Documentation

### 8.9.1 void CRC\_Init ( **CRC\_Type** \* *base*, **const crc\_config\_t** \* *config* )

This function enables the clock gate in the SIM module for the CRC peripheral. It also configures the CRC module and starts a checksum computation by writing the seed.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | CRC peripheral address.             |
| <i>config</i> | CRC module configuration structure. |

### 8.9.2 static void CRC\_Deinit ( **CRC\_Type** \* *base* ) [inline], [static]

This function disables the clock gate in the SIM module for the CRC peripheral.

Parameters

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

### 8.9.3 void CRC\_GetDefaultConfig ( **crc\_config\_t** \* *config* )

Loads default values to the CRC protocol configuration structure. The default values are as follows.

```
* config->polynomial = 0x1021;
* config->seed = 0xFFFF;
* config->reflectIn = false;
* config->reflectOut = false;
* config->complementChecksum = false;
* config->crcBits = kCrcBits16;
* config->crcResult = kCrcFinalChecksum;
*
```

Parameters

|               |                                       |
|---------------|---------------------------------------|
| <i>config</i> | CRC protocol configuration structure. |
|---------------|---------------------------------------|

### 8.9.4 void CRC\_WriteData ( **CRC\_Type** \* *base*, **const uint8\_t** \* *data*, **size\_t** *dataSize* )

Writes input data buffer bytes to the CRC data register. The configured type of transpose is applied.

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | CRC peripheral address.                 |
| <i>data</i>     | Input data stream, MSByte in data[0].   |
| <i>dataSize</i> | Size in bytes of the input data buffer. |

### 8.9.5 `uint32_t CRC_Get32bitResult ( CRC_Type * base )`

Reads the CRC data register (either an intermediate or the final checksum). The configured type of transpose and complement is applied.

Parameters

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

Returns

An intermediate or the final 32-bit checksum, after configured transpose and complement operations.

### 8.9.6 `uint16_t CRC_Get16bitResult ( CRC_Type * base )`

Reads the CRC data register (either an intermediate or the final checksum). The configured type of transpose and complement is applied.

Parameters

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

Returns

An intermediate or the final 16-bit checksum, after configured transpose and complement operations.

# Chapter 9

## DAC: Digital-to-Analog Converter Driver

### 9.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Digital-to-Analog Converter (DAC) module of MCUXpresso SDK devices.

The DAC driver includes a basic DAC module (converter) and a DAC buffer.

The basic DAC module supports operations unique to the DAC converter in each DAC instance. The APIs in this section are used in the initialization phase, which enables the DAC module in the application. The APIs enable/disable the clock, enable/disable the module, and configure the converter. Call the initial APIs to prepare the DAC module for the application.

The DAC buffer operates the DAC hardware buffer. The DAC module supports a hardware buffer to keep a group of DAC values to be converted. This feature supports updating the DAC output value automatically by triggering the buffer read pointer to move in the buffer. Use the APIs to configure the hardware buffer's trigger mode, watermark, work mode, and use size. Additionally, the APIs operate the DMA, interrupts, flags, the pointer (the index of the buffer), item values, and so on.

Note that the most functional features are designed for the DAC hardware buffer.

### 9.2 Typical use case

#### 9.2.1 Working as a basic DAC without the hardware buffer feature

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/dac

#### 9.2.2 Working with the hardware buffer

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/dac

### Data Structures

- struct `dac_config_t`  
*DAC module configuration.* [More...](#)
- struct `dac_buffer_config_t`  
*DAC buffer configuration.* [More...](#)

### Enumerations

- enum `_dac_buffer_status_flags` {  
  `kDAC_BufferWatermarkFlag` = DAC\_SR\_DACBFWMF\_MASK,  
  `kDAC_BufferReadPointerTopPositionFlag` = DAC\_SR\_DACBFRPTF\_MASK,

- ```

kDAC_BufferReadPointerBottomPositionFlag = DAC_SR_DACBFRPBF_MASK }

DAC buffer flags.
• enum _dac_buffer_interrupt_enable {
    kDAC_BufferWatermarkInterruptEnable = DAC_C0_DACBWIEN_MASK,
    kDAC_BufferReadPointerTopInterruptEnable = DAC_C0_DACBTIEN_MASK,
    kDAC_BufferReadPointerBottomInterruptEnable = DAC_C0_DACBBIEN_MASK }

DAC buffer interrupts.
• enum dac_reference_voltage_source_t {
    kDAC_ReferenceVoltageSourceVref1 = 0U,
    kDAC_ReferenceVoltageSourceVref2 = 1U }

DAC reference voltage source.
• enum dac_buffer_trigger_mode_t {
    kDAC_BufferTriggerByHardwareMode = 0U,
    kDAC_BufferTriggerBySoftwareMode = 1U }

DAC buffer trigger mode.
• enum dac_buffer_watermark_t {
    kDAC_BufferWatermark1Word = 0U,
    kDAC_BufferWatermark2Word = 1U,
    kDAC_BufferWatermark3Word = 2U,
    kDAC_BufferWatermark4Word = 3U }

DAC buffer watermark.
• enum dac_buffer_work_mode_t {
    kDAC_BufferWorkAsNormalMode = 0U,
    kDAC_BufferWorkAsSwingMode,
    kDAC_BufferWorkAsOneTimeScanMode,
    kDAC_BufferWorkAsFIFOMode }

DAC buffer work mode.

```

Driver version

- #define **FSL_DAC_DRIVER_VERSION** (MAKE_VERSION(2, 0, 2))
 DAC driver version 2.0.2.

Initialization

- void **DAC_Init** (DAC_Type *base, const **dac_config_t** *config)
 Initializes the DAC module.
- void **DAC_Deinit** (DAC_Type *base)
 De-initializes the DAC module.
- void **DAC_GetDefaultConfig** (**dac_config_t** *config)
 Initializes the DAC user configuration structure.
- static void **DAC_Enable** (DAC_Type *base, bool enable)
 Enables the DAC module.

Buffer

- static void **DAC_EnableBuffer** (DAC_Type *base, bool enable)
 Enables the DAC buffer.

- void [DAC_SetBufferConfig](#) (DAC_Type *base, const [dac_buffer_config_t](#) *config)
Configures the CMP buffer.
- void [DAC_GetDefaultBufferConfig](#) ([dac_buffer_config_t](#) *config)
Initializes the DAC buffer configuration structure.
- static void [DAC_EnableBufferDMA](#) (DAC_Type *base, bool enable)
Enables the DMA for DAC buffer.
- void [DAC_SetBufferValue](#) (DAC_Type *base, uint8_t index, uint16_t value)
Sets the value for items in the buffer.
- static void [DAC_DoSoftwareTriggerBuffer](#) (DAC_Type *base)
Triggers the buffer using software and updates the read pointer of the DAC buffer.
- static uint8_t [DAC_GetBufferReadPointer](#) (DAC_Type *base)
Gets the current read pointer of the DAC buffer.
- void [DAC_SetBufferReadPointer](#) (DAC_Type *base, uint8_t index)
Sets the current read pointer of the DAC buffer.
- void [DAC_EnableBufferInterrupts](#) (DAC_Type *base, uint32_t mask)
Enables interrupts for the DAC buffer.
- void [DAC_DisableBufferInterrupts](#) (DAC_Type *base, uint32_t mask)
Disables interrupts for the DAC buffer.
- uint8_t [DAC_GetBufferStatusFlags](#) (DAC_Type *base)
Gets the flags of events for the DAC buffer.
- void [DAC_ClearBufferStatusFlags](#) (DAC_Type *base, uint32_t mask)
Clears the flags of events for the DAC buffer.

9.3 Data Structure Documentation

9.3.1 struct [dac_config_t](#)

Data Fields

- [dac_reference_voltage_source_t](#) [referenceVoltageSource](#)
Select the DAC reference voltage source.
- bool [enableLowPowerMode](#)
Enable the low-power mode.

Field Documentation

- (1) [dac_reference_voltage_source_t](#) [dac_config_t::referenceVoltageSource](#)
- (2) bool [dac_config_t::enableLowPowerMode](#)

9.3.2 struct [dac_buffer_config_t](#)

Data Fields

- [dac_buffer_trigger_mode_t](#) [triggerMode](#)
Select the buffer's trigger mode.
- [dac_buffer_watermark_t](#) [watermark](#)
Select the buffer's watermark.
- [dac_buffer_work_mode_t](#) [workMode](#)

- *Select the buffer's work mode.*
- `uint8_t upperLimit`
Set the upper limit for the buffer index.

Field Documentation

- (1) `dac_buffer_trigger_mode_t dac_buffer_config_t::triggerMode`
- (2) `dac_buffer_watermark_t dac_buffer_config_t::watermark`
- (3) `dac_buffer_work_mode_t dac_buffer_config_t::workMode`
- (4) `uint8_t dac_buffer_config_t::upperLimit`

Normally, 0-15 is available for a buffer with 16 items.

9.4 Macro Definition Documentation

9.4.1 #define FSL_DAC_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

9.5 Enumeration Type Documentation

9.5.1 enum _dac_buffer_status_flags

Enumerator

kDAC_BufferWatermarkFlag DAC Buffer Watermark Flag.

kDAC_BufferReadPointerTopPositionFlag DAC Buffer Read Pointer Top Position Flag.

kDAC_BufferReadPointerBottomPositionFlag DAC Buffer Read Pointer Bottom Position Flag.

9.5.2 enum _dac_buffer_interrupt_enable

Enumerator

kDAC_BufferWatermarkInterruptEnable DAC Buffer Watermark Interrupt Enable.

kDAC_BufferReadPointerTopInterruptEnable DAC Buffer Read Pointer Top Flag Interrupt Enable.

kDAC_BufferReadPointerBottomInterruptEnable DAC Buffer Read Pointer Bottom Flag Interrupt Enable.

9.5.3 enum dac_reference_voltage_source_t

Enumerator

kDAC_ReferenceVoltageSourceVref1 The DAC selects DACREF_1 as the reference voltage.

kDAC_ReferenceVoltageSourceVref2 The DAC selects DACREF_2 as the reference voltage.

9.5.4 enum dac_buffer_trigger_mode_t

Enumerator

kDAC_BufferTriggerByHardwareMode The DAC hardware trigger is selected.

kDAC_BufferTriggerBySoftwareMode The DAC software trigger is selected.

9.5.5 enum dac_buffer_watermark_t

Enumerator

kDAC_BufferWatermark1Word 1 word away from the upper limit.

kDAC_BufferWatermark2Word 2 words away from the upper limit.

kDAC_BufferWatermark3Word 3 words away from the upper limit.

kDAC_BufferWatermark4Word 4 words away from the upper limit.

9.5.6 enum dac_buffer_work_mode_t

Enumerator

kDAC_BufferWorkAsNormalMode Normal mode.

kDAC_BufferWorkAsSwingMode Swing mode.

kDAC_BufferWorkAsOneTimeScanMode One-Time Scan mode.

kDAC_BufferWorkAsFIFOMode FIFO mode.

9.6 Function Documentation

9.6.1 void DAC_Init(DAC_Type * *base*, const dac_config_t * *config*)

This function initializes the DAC module including the following operations.

- Enabling the clock for DAC module.
- Configuring the DAC converter with a user configuration.
- Enabling the DAC module.

Parameters

<i>base</i>	DAC peripheral base address.
-------------	------------------------------

<i>config</i>	Pointer to the configuration structure. See "dac_config_t".
---------------	---

9.6.2 void DAC_Deinit (DAC_Type * *base*)

This function de-initializes the DAC module including the following operations.

- Disabling the DAC module.
- Disabling the clock for the DAC module.

Parameters

<i>base</i>	DAC peripheral base address.
-------------	------------------------------

9.6.3 void DAC_GetDefaultConfig (dac_config_t * *config*)

This function initializes the user configuration structure to a default value. The default values are as follows.

```
* config->referenceVoltageSource = kDAC_ReferenceVoltageSourceVref2;
* config->enableLowPowerMode = false;
*
```

Parameters

<i>config</i>	Pointer to the configuration structure. See "dac_config_t".
---------------	---

9.6.4 static void DAC_Enable (DAC_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	DAC peripheral base address.
<i>enable</i>	Enables or disables the feature.

9.6.5 static void DAC_EnableBuffer (DAC_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	DAC peripheral base address.
<i>enable</i>	Enables or disables the feature.

9.6.6 void DAC_SetBufferConfig (DAC_Type * *base*, const dac_buffer_config_t * *config*)

Parameters

<i>base</i>	DAC peripheral base address.
<i>config</i>	Pointer to the configuration structure. See "dac_buffer_config_t".

9.6.7 void DAC_GetDefaultBufferConfig (dac_buffer_config_t * *config*)

This function initializes the DAC buffer configuration structure to default values. The default values are as follows.

```
* config->triggerMode = kDAC_BufferTriggerBySoftwareMode;
* config->watermark   = kDAC_BufferWatermark1Word;
* config->workMode    = kDAC_BufferWorkAsNormalMode;
* config->upperLimit  = DAC_DATL_COUNT - 1U;
*
```

Parameters

<i>config</i>	Pointer to the configuration structure. See "dac_buffer_config_t".
---------------	--

9.6.8 static void DAC_EnableBufferDMA (DAC_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	DAC peripheral base address.
-------------	------------------------------

<i>enable</i>	Enables or disables the feature.
---------------	----------------------------------

9.6.9 void DAC_SetBufferValue (**DAC_Type** * *base*, **uint8_t** *index*, **uint16_t** *value*)

Parameters

<i>base</i>	DAC peripheral base address.
<i>index</i>	Setting the index for items in the buffer. The available index should not exceed the size of the DAC buffer.
<i>value</i>	Setting the value for items in the buffer. 12-bits are available.

9.6.10 static void DAC_DoSoftwareTriggerBuffer (**DAC_Type** * *base*) [inline], [static]

This function triggers the function using software. The read pointer of the DAC buffer is updated with one step after this function is called. Changing the read pointer depends on the buffer's work mode.

Parameters

<i>base</i>	DAC peripheral base address.
-------------	------------------------------

9.6.11 static uint8_t DAC_GetBufferReadPointer (**DAC_Type** * *base*) [inline], [static]

This function gets the current read pointer of the DAC buffer. The current output value depends on the item indexed by the read pointer. It is updated either by a software trigger or a hardware trigger.

Parameters

<i>base</i>	DAC peripheral base address.
-------------	------------------------------

Returns

The current read pointer of the DAC buffer.

9.6.12 void DAC_SetBufferReadPointer (DAC_Type * *base*, uint8_t *index*)

This function sets the current read pointer of the DAC buffer. The current output value depends on the item indexed by the read pointer. It is updated either by a software trigger or a hardware trigger. After the read pointer changes, the DAC output value also changes.

Parameters

<i>base</i>	DAC peripheral base address.
<i>index</i>	Setting an index value for the pointer.

9.6.13 void DAC_EnableBufferInterrupts (DAC_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	DAC peripheral base address.
<i>mask</i>	Mask value for interrupts. See "_dac_buffer_interrupt_enable".

9.6.14 void DAC_DisableBufferInterrupts (DAC_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	DAC peripheral base address.
<i>mask</i>	Mask value for interrupts. See "_dac_buffer_interrupt_enable".

9.6.15 uint8_t DAC_GetBufferStatusFlags (DAC_Type * *base*)

Parameters

<i>base</i>	DAC peripheral base address.
-------------	------------------------------

Returns

Mask value for the asserted flags. See "_dac_buffer_status_flags".

9.6.16 void DAC_ClearBufferStatusFlags (DAC_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	DAC peripheral base address.
<i>mask</i>	Mask value for flags. See "_dac_buffer_status_flags_t".

Chapter 10

DMAMUX: Direct Memory Access Multiplexer Driver

10.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Direct Memory Access Multiplexer (DMAMUX) of MCUXpresso SDK devices.

10.2 Typical use case

10.2.1 DMAMUX Operation

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

Driver version

- #define `FSL_DMAMUX_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 5)`)
DMAMUX driver version 2.0.5.

DMAMUX Initialization and de-initialization

- void `DMAMUX_Init` (DMAMUX_Type *base)
Initializes the DMAMUX peripheral.
- void `DMAMUX_Deinit` (DMAMUX_Type *base)
Deinitializes the DMAMUX peripheral.

DMAMUX Channel Operation

- static void `DMAMUX_EnableChannel` (DMAMUX_Type *base, uint32_t channel)
Enables the DMAMUX channel.
- static void `DMAMUX_DisableChannel` (DMAMUX_Type *base, uint32_t channel)
Disables the DMAMUX channel.
- static void `DMAMUX_SetSource` (DMAMUX_Type *base, uint32_t channel, uint32_t source)
Configures the DMAMUX channel source.
- static void `DMAMUX_EnablePeriodTrigger` (DMAMUX_Type *base, uint32_t channel)
Enables the DMAMUX period trigger.
- static void `DMAMUX_DisablePeriodTrigger` (DMAMUX_Type *base, uint32_t channel)
Disables the DMAMUX period trigger.

10.3 Macro Definition Documentation

10.3.1 #define `FSL_DMAMUX_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 5)`)

10.4 Function Documentation

10.4.1 void DMAMUX_Init (**DMAMUX_Type** * *base*)

This function ungates the DMAMUX clock.

Parameters

<i>base</i>	DMAMUX peripheral base address.
-------------	---------------------------------

10.4.2 void DMAMUX_Deinit (DMAMUX_Type * *base*)

This function gates the DMAMUX clock.

Parameters

<i>base</i>	DMAMUX peripheral base address.
-------------	---------------------------------

10.4.3 static void DMAMUX_EnableChannel (DMAMUX_Type * *base*, uint32_t *channel*) [inline], [static]

This function enables the DMAMUX channel.

Parameters

<i>base</i>	DMAMUX peripheral base address.
<i>channel</i>	DMAMUX channel number.

10.4.4 static void DMAMUX_DisableChannel (DMAMUX_Type * *base*, uint32_t *channel*) [inline], [static]

This function disables the DMAMUX channel.

Note

The user must disable the DMAMUX channel before configuring it.

Parameters

<i>base</i>	DMAMUX peripheral base address.
-------------	---------------------------------

<i>channel</i>	DMAMUX channel number.
----------------	------------------------

10.4.5 static void DMAMUX_SetSource (DMAMUX_Type * *base*, uint32_t *channel*, uint32_t *source*) [inline], [static]

Parameters

<i>base</i>	DMAMUX peripheral base address.
<i>channel</i>	DMAMUX channel number.
<i>source</i>	Channel source, which is used to trigger the DMA transfer.

10.4.6 static void DMAMUX_EnablePeriodTrigger (DMAMUX_Type * *base*, uint32_t *channel*) [inline], [static]

This function enables the DMAMUX period trigger feature.

Parameters

<i>base</i>	DMAMUX peripheral base address.
<i>channel</i>	DMAMUX channel number.

10.4.7 static void DMAMUX_DisablePeriodTrigger (DMAMUX_Type * *base*, uint32_t *channel*) [inline], [static]

This function disables the DMAMUX period trigger.

Parameters

<i>base</i>	DMAMUX peripheral base address.
<i>channel</i>	DMAMUX channel number.

Chapter 11

DSPI: Serial Peripheral Interface Driver

11.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Serial Peripheral Interface (SPI) module of MCUXpresso SDK devices.

Modules

- DSPI CMSIS Driver
- DSPI Driver
- DSPI FreeRTOS Driver
- DSPI eDMA Driver

11.2 DSPI Driver

11.2.1 Overview

This section describes the programming interface of the DSPI peripheral driver. The DSPI driver configures the DSPI module and provides functional and transactional interfaces to build the DSPI application.

11.2.2 Typical use case

11.2.2.1 Master Operation

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

11.2.2.2 Slave Operation

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

Data Structures

- struct `dspi_command_data_config_t`
DSPI master command date configuration used for the SPIx_PUSHR. [More...](#)
- struct `dspi_master_ctar_config_t`
DSPI master ctar configuration structure. [More...](#)
- struct `dspi_master_config_t`
DSPI master configuration structure. [More...](#)
- struct `dspi_slave_ctar_config_t`
DSPI slave ctar configuration structure. [More...](#)
- struct `dspi_slave_config_t`
DSPI slave configuration structure. [More...](#)
- struct `dspi_transfer_t`
DSPI master/slave transfer structure. [More...](#)
- struct `dspi_half_duplex_transfer_t`
DSPI half-duplex(master) transfer structure. [More...](#)
- struct `dspi_master_handle_t`
DSPI master transfer handle structure used for transactional API. [More...](#)
- struct `dspi_slave_handle_t`
DSPI slave transfer handle structure used for the transactional API. [More...](#)

Macros

- #define `DSPI_DUMMY_DATA` (0x00U)
DSPI dummy data if there is no Tx data.
- #define `DSPI_MASTER_CTAR_SHIFT` (0U)
DSPI master CTAR shift macro; used internally.

- #define **DSPI_MASTER_CTAR_MASK** (0x0FU)
DSPI master CTAR mask macro; used internally.
- #define **DSPI_MASTER_PCS_SHIFT** (4U)
DSPI master PCS shift macro; used internally.
- #define **DSPI_MASTER_PCS_MASK** (0xF0U)
DSPI master PCS mask macro; used internally.
- #define **DSPI_SLAVE_CTAR_SHIFT** (0U)
DSPI slave CTAR shift macro; used internally.
- #define **DSPI_SLAVE_CTAR_MASK** (0x07U)
DSPI slave CTAR mask macro; used internally.

Typedefs

- typedef void(* **dspi_master_transfer_callback_t**)(SPI_Type *base, dspi_master_handle_t *handle, **status_t** status, void *userData)
Completion callback function pointer type.
- typedef void(* **dspi_slave_transfer_callback_t**)(SPI_Type *base, dspi_slave_handle_t *handle, **status_t** status, void *userData)
Completion callback function pointer type.

Enumerations

- enum {
 kStatus_DSPI_Busy = MAKE_STATUS(kStatusGroup_DSPI, 0),
 kStatus_DSPI_Error = MAKE_STATUS(kStatusGroup_DSPI, 1),
 kStatus_DSPI_Idle = MAKE_STATUS(kStatusGroup_DSPI, 2),
 kStatus_DSPI_OutOfRange = MAKE_STATUS(kStatusGroup_DSPI, 3) }

Status for the DSPI driver.
- enum **_dspi_flags** {
 kDSPI_TxCompleteFlag = (int)SPI_SR_TCF_MASK,
 kDSPI_EndOfQueueFlag = SPI_SR_EOQF_MASK,
 kDSPI_TxFifoUnderflowFlag = SPI_SR_TFUF_MASK,
 kDSPI_TxFifoFillRequestFlag = SPI_SR_TFFF_MASK,
 kDSPI_RxFifoOverflowFlag = SPI_SR_RFOF_MASK,
 kDSPI_RxFifoDrainRequestFlag = SPI_SR_RFDF_MASK,
 kDSPI_TxAndRxStatusFlag = SPI_SR_TXRXS_MASK,
 kDSPI_AllStatusFlag }

DSPI status flags in SPIx_SR register.
- enum **_dspi_interrupt_enable** {
 kDSPI_TxCompleteInterruptEnable = (int)SPI_RSER_TCF_RE_MASK,
 kDSPI_EndOfQueueInterruptEnable = SPI_RSER_EOQF_RE_MASK,
 kDSPI_TxFifoUnderflowInterruptEnable = SPI_RSER_TFUF_RE_MASK,
 kDSPI_TxFifoFillRequestInterruptEnable = SPI_RSER_TFFF_RE_MASK,
 kDSPI_RxFifoOverflowInterruptEnable = SPI_RSER_RFOF_RE_MASK,
 kDSPI_RxFifoDrainRequestInterruptEnable = SPI_RSER_RFDF_RE_MASK,

- ```
kDSPI_AllInterruptEnable }
```

*DSPI interrupt source.*
- enum `_dspi_dma_enable` {
   
`kDSPI_TxDmaEnable` = (SPI\_RSER\_TFFF\_RE\_MASK | SPI\_RSER\_TFFF\_DIRS\_MASK),  
`kDSPI_RxDmaEnable` = (SPI\_RSER\_RFDF\_RE\_MASK | SPI\_RSER\_RFDF\_DIRS\_MASK) }

*DSPI DMA source.*
- enum `dspi_master_slave_mode_t` {
   
`kDSPI_Master` = 1U,  
`kDSPI_Slave` = 0U }

*DSPI master or slave mode configuration.*
- enum `dspi_master_sample_point_t` {
   
`kDSPI_SckToSin0Clock` = 0U,  
`kDSPI_SckToSin1Clock` = 1U,  
`kDSPI_SckToSin2Clock` = 2U }

*DSPI Sample Point: Controls when the DSPI master samples SIN in the Modified Transfer Format.*
- enum `dspi_which_pcs_t` {
   
`kDSPI_Pcs0` = 1U << 0,  
`kDSPI_Pcs1` = 1U << 1,  
`kDSPI_Pcs2` = 1U << 2,  
`kDSPI_Pcs3` = 1U << 3,  
`kDSPI_Pcs4` = 1U << 4,  
`kDSPI_Pcs5` = 1U << 5 }

*DSPI Peripheral Chip Select (Pcs) configuration (which Pcs to configure).*
- enum `dspi_pcs_polarity_config_t` {
   
`kDSPI_PcsActiveHigh` = 0U,  
`kDSPI_PcsActiveLow` = 1U }

*DSPI Peripheral Chip Select (Pcs) Polarity configuration.*
- enum `_dspi_pcs_polarity` {
   
`kDSPI_Pcs0ActiveLow` = 1U << 0,  
`kDSPI_Pcs1ActiveLow` = 1U << 1,  
`kDSPI_Pcs2ActiveLow` = 1U << 2,  
`kDSPI_Pcs3ActiveLow` = 1U << 3,  
`kDSPI_Pcs4ActiveLow` = 1U << 4,  
`kDSPI_Pcs5ActiveLow` = 1U << 5,  
`kDSPI_PcsAllActiveLow` = 0xFFU }

*DSPI Peripheral Chip Select (Pcs) Polarity.*
- enum `dspi_clock_polarity_t` {
   
`kDSPI_ClockPolarityActiveHigh` = 0U,  
`kDSPI_ClockPolarityActiveLow` = 1U }

*DSPI clock polarity configuration for a given CTAR.*
- enum `dspi_clock_phase_t` {
   
`kDSPI_ClockPhaseFirstEdge` = 0U,  
`kDSPI_ClockPhaseSecondEdge` = 1U }

*DSPI clock phase configuration for a given CTAR.*
- enum `dspi_shift_direction_t` {
   
`kDSPI_MsbFirst` = 0U,  
`kDSPI_LsbFirst` = 1U }

*DSPI data shifter direction options for a given CTAR.*

- enum `dspi_delay_type_t` {
   
    `kDSPI_PesToSck` = 1U,
   
    `kDSPI_LastSckToPcs`,
   
    `kDSPI_BetweenTransfer` }
- DSPI delay type selection.*
- enum `dspi_ctar_selection_t` {
   
    `kDSPI_Ctar0` = 0U,
   
    `kDSPI_Ctar1` = 1U,
   
    `kDSPI_Ctar2` = 2U,
   
    `kDSPI_Ctar3` = 3U,
   
    `kDSPI_Ctar4` = 4U,
   
    `kDSPI_Ctar5` = 5U,
   
    `kDSPI_Ctar6` = 6U,
   
    `kDSPI_Ctar7` = 7U }

*DSPI Clock and Transfer Attributes Register (CTAR) selection.*

- enum `_dspi_transfer_config_flag_for_master` {
   
    `kDSPI_MasterCtar0` = 0U << DSPI\_MASTER\_CTAR\_SHIFT,
   
    `kDSPI_MasterCtar1` = 1U << DSPI\_MASTER\_CTAR\_SHIFT,
   
    `kDSPI_MasterCtar2` = 2U << DSPI\_MASTER\_CTAR\_SHIFT,
   
    `kDSPI_MasterCtar3` = 3U << DSPI\_MASTER\_CTAR\_SHIFT,
   
    `kDSPI_MasterCtar4` = 4U << DSPI\_MASTER\_CTAR\_SHIFT,
   
    `kDSPI_MasterCtar5` = 5U << DSPI\_MASTER\_CTAR\_SHIFT,
   
    `kDSPI_MasterCtar6` = 6U << DSPI\_MASTER\_CTAR\_SHIFT,
   
    `kDSPI_MasterCtar7` = 7U << DSPI\_MASTER\_CTAR\_SHIFT,
   
    `kDSPI_MasterPcs0` = 0U << DSPI\_MASTER\_PCS\_SHIFT,
   
    `kDSPI_MasterPcs1` = 1U << DSPI\_MASTER\_PCS\_SHIFT,
   
    `kDSPI_MasterPcs2` = 2U << DSPI\_MASTER\_PCS\_SHIFT,
   
    `kDSPI_MasterPcs3` = 3U << DSPI\_MASTER\_PCS\_SHIFT,
   
    `kDSPI_MasterPcs4` = 4U << DSPI\_MASTER\_PCS\_SHIFT,
   
    `kDSPI_MasterPcs5` = 5U << DSPI\_MASTER\_PCS\_SHIFT,
   
    `kDSPI_MasterPcsContinuous` = 1U << 20,
   
    `kDSPI_MasterActiveAfterTransfer` = 1U << 21 }

*Use this enumeration for the DSPI master transfer configFlags.*

- enum `_dspi_transfer_config_flag_for_slave` { `kDSPI_SlaveCtar0` = 0U << DSPI\_SLAVE\_CTAR\_SHIFT }

*Use this enumeration for the DSPI slave transfer configFlags.*

- enum `_dspi_transfer_state` {
   
    `kDSPI_Idle` = 0x0U,
   
    `kDSPI_Busy`,
   
    `kDSPI_Error` }

*DSPI transfer state, which is used for DSPI transactional API state machine.*

## Variables

- volatile uint8\_t `g_dspiDummyData []`  
*Global variable for dummy data value setting.*

## Driver version

- #define `FSL_DSPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 4))`  
*DSPI driver version 2.2.4.*

## Initialization and deinitialization

- void `DSPI_MasterInit` (SPI\_Type \*base, const `dspi_master_config_t` \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes the DSPI master.*
- void `DSPI_MasterGetDefaultConfig` (`dspi_master_config_t` \*masterConfig)  
*Sets the `dspi_master_config_t` structure to default values.*
- void `DSPI_SlaveInit` (SPI\_Type \*base, const `dspi_slave_config_t` \*slaveConfig)  
*DSPI slave configuration.*
- void `DSPI_SlaveGetDefaultConfig` (`dspi_slave_config_t` \*slaveConfig)  
*Sets the `dspi_slave_config_t` structure to a default value.*
- void `DSPI_Deinit` (SPI\_Type \*base)  
*De-initializes the DSPI peripheral.*
- static void `DSPI_Enable` (SPI\_Type \*base, bool enable)  
*Enables the DSPI peripheral and sets the MCR MDIS to 0.*

## Status

- static uint32\_t `DSPI_GetStatusFlags` (SPI\_Type \*base)  
*Gets the DSPI status flag state.*
- static void `DSPI_ClearStatusFlags` (SPI\_Type \*base, uint32\_t statusFlags)  
*Clears the DSPI status flag.*

## Interrupts

- void `DSPI_EnableInterrupts` (SPI\_Type \*base, uint32\_t mask)  
*Enables the DSPI interrupts.*
- static void `DSPI_DisableInterrupts` (SPI\_Type \*base, uint32\_t mask)  
*Disables the DSPI interrupts.*

## DMA Control

- static void `DSPI_EnableDMA` (SPI\_Type \*base, uint32\_t mask)  
*Enables the DSPI DMA request.*

- static void **DSPI\_DisableDMA** (SPI\_Type \*base, uint32\_t mask)  
*Disables the DSPI DMA request.*
- static uint32\_t **DSPI\_MasterGetTxRegisterAddress** (SPI\_Type \*base)  
*Gets the DSPI master PUSHR data register address for the DMA operation.*
- static uint32\_t **DSPI\_SlaveGetTxRegisterAddress** (SPI\_Type \*base)  
*Gets the DSPI slave PUSHR data register address for the DMA operation.*
- static uint32\_t **DSPI\_GetRxRegisterAddress** (SPI\_Type \*base)  
*Gets the DSPI POPR data register address for the DMA operation.*

## Bus Operations

- uint32\_t **DSPIGetInstance** (SPI\_Type \*base)  
*Get instance number for DSPI module.*
- static void **DSPI\_SetMasterSlaveMode** (SPI\_Type \*base, **dspi\_master\_slave\_mode\_t** mode)  
*Configures the DSPI for master or slave.*
- static bool **DSPI\_IsMaster** (SPI\_Type \*base)  
*Returns whether the DSPI module is in master mode.*
- static void **DSPI\_StartTransfer** (SPI\_Type \*base)  
*Starts the DSPI transfers and clears HALT bit in MCR.*
- static void **DSPI\_StopTransfer** (SPI\_Type \*base)  
*Stops DSPI transfers and sets the HALT bit in MCR.*
- static void **DSPI\_SetFifoEnable** (SPI\_Type \*base, bool enableTxFifo, bool enableRxFifo)  
*Enables or disables the DSPI FIFOs.*
- static void **DSPI\_FlushFifo** (SPI\_Type \*base, bool flushTxFifo, bool flushRxFifo)  
*Flushes the DSPI FIFOs.*
- static void **DSPI\_SetAllPcsPolarity** (SPI\_Type \*base, uint32\_t mask)  
*Configures the DSPI peripheral chip select polarity simultaneously.*
- uint32\_t **DSPI\_MasterSetBaudRate** (SPI\_Type \*base, **dspi\_ctar\_selection\_t** whichCtar, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the DSPI baud rate in bits per second.*
- void **DSPI\_MasterSetDelayScaler** (SPI\_Type \*base, **dspi\_ctar\_selection\_t** whichCtar, uint32\_t prescaler, uint32\_t scaler, **dspi\_delay\_type\_t** whichDelay)  
*Manually configures the delay prescaler and scaler for a particular CTAR.*
- uint32\_t **DSPI\_MasterSetDelayTimes** (SPI\_Type \*base, **dspi\_ctar\_selection\_t** whichCtar, **dspi\_delay\_type\_t** whichDelay, uint32\_t srcClock\_Hz, uint32\_t delayTimeInNanoSec)  
*Calculates the delay prescaler and scaler based on the desired delay input in nanoseconds.*
- static void **DSPI\_MasterWriteData** (SPI\_Type \*base, **dspi\_command\_data\_config\_t** \*command, uint16\_t data)  
*Writes data into the data buffer for master mode.*
- void **DSPI\_GetDefaultDataCommandConfig** (**dspi\_command\_data\_config\_t** \*command)  
*Sets the **dspi\_command\_data\_config\_t** structure to default values.*
- void **DSPI\_MasterWriteDataBlocking** (SPI\_Type \*base, **dspi\_command\_data\_config\_t** \*command, uint16\_t data)  
*Writes data into the data buffer master mode and waits till complete to return.*
- static uint32\_t **DSPI\_MasterGetFormattedCommand** (**dspi\_command\_data\_config\_t** \*command)  
*Returns the DSPI command word formatted to the PUSHR data register bit field.*
- void **DSPI\_MasterWriteCommandDataBlocking** (SPI\_Type \*base, uint32\_t data)  
*Writes a 32-bit data word (16-bit command appended with 16-bit data) into the data buffer master mode and waits till complete to return.*

- static void **DSPI\_SlaveWriteData** (SPI\_Type \*base, uint32\_t data)  
*Writes data into the data buffer in slave mode.*
- void **DSPI\_SlaveWriteDataBlocking** (SPI\_Type \*base, uint32\_t data)  
*Writes data into the data buffer in slave mode, waits till data was transmitted, and returns.*
- static uint32\_t **DSPI\_ReadData** (SPI\_Type \*base)  
*Reads data from the data buffer.*
- void **DSPI\_SetDummyData** (SPI\_Type \*base, uint8\_t dummyData)  
*Set up the dummy data.*

## Transactional APIs

- void **DSPI\_MasterTransferCreateHandle** (SPI\_Type \*base, dspi\_master\_handle\_t \*handle, **dspi\_master\_transfer\_callback\_t** callback, void \*userData)  
*Initializes the DSPI master handle.*
- status\_t **DSPI\_MasterTransferBlocking** (SPI\_Type \*base, **dspi\_transfer\_t** \*transfer)  
*DSPI master transfer data using polling.*
- status\_t **DSPI\_MasterTransferNonBlocking** (SPI\_Type \*base, dspi\_master\_handle\_t \*handle, **dspi\_transfer\_t** \*transfer)  
*DSPI master transfer data using interrupts.*
- status\_t **DSPI\_MasterHalfDuplexTransferBlocking** (SPI\_Type \*base, **dspi\_half\_duplex\_transfer\_t** \*xfer)  
*Transfers a block of data using a polling method.*
- status\_t **DSPI\_MasterHalfDuplexTransferNonBlocking** (SPI\_Type \*base, dspi\_master\_handle\_t \*handle, **dspi\_half\_duplex\_transfer\_t** \*xfer)  
*Performs a non-blocking DSPI interrupt transfer.*
- status\_t **DSPI\_MasterTransferGetCount** (SPI\_Type \*base, dspi\_master\_handle\_t \*handle, size\_t \*count)  
*Gets the master transfer count.*
- void **DSPI\_MasterTransferAbort** (SPI\_Type \*base, dspi\_master\_handle\_t \*handle)  
*DSPI master aborts a transfer using an interrupt.*
- void **DSPI\_MasterTransferHandleIRQ** (SPI\_Type \*base, dspi\_master\_handle\_t \*handle)  
*DSPI Master IRQ handler function.*
- void **DSPI\_SlaveTransferCreateHandle** (SPI\_Type \*base, dspi\_slave\_handle\_t \*handle, **dspi\_slave\_transfer\_callback\_t** callback, void \*userData)  
*Initializes the DSPI slave handle.*
- status\_t **DSPI\_SlaveTransferNonBlocking** (SPI\_Type \*base, dspi\_slave\_handle\_t \*handle, **dspi\_transfer\_t** \*transfer)  
*DSPI slave transfers data using an interrupt.*
- status\_t **DSPI\_SlaveTransferGetCount** (SPI\_Type \*base, dspi\_slave\_handle\_t \*handle, size\_t \*count)  
*Gets the slave transfer count.*
- void **DSPI\_SlaveTransferAbort** (SPI\_Type \*base, dspi\_slave\_handle\_t \*handle)  
*DSPI slave aborts a transfer using an interrupt.*
- void **DSPI\_SlaveTransferHandleIRQ** (SPI\_Type \*base, dspi\_slave\_handle\_t \*handle)  
*DSPI Master IRQ handler function.*
- uint8\_t **DSPI\_GetDummyDataInstance** (SPI\_Type \*base)  
*brief Dummy data for each instance.*

## 11.2.3 Data Structure Documentation

### 11.2.3.1 struct dspi\_command\_data\_config\_t

#### Data Fields

- bool [isPcsContinuous](#)  
*Option to enable the continuous assertion of the chip select between transfers.*
- uint8\_t [whichCtar](#)  
*The desired Clock and Transfer Attributes Register (CTAR) to use for CTAS.*
- uint8\_t [whichPcs](#)  
*The desired PCS signal to use for the data transfer.*
- bool [isEndOfQueue](#)  
*Signals that the current transfer is the last in the queue.*
- bool [clearTransferCount](#)  
*Clears the SPI Transfer Counter (SPI\_TCNT) before transmission starts.*

#### Field Documentation

- (1) bool [dspi\\_command\\_data\\_config\\_t::isPcsContinuous](#)
- (2) uint8\_t [dspi\\_command\\_data\\_config\\_t::whichCtar](#)
- (3) uint8\_t [dspi\\_command\\_data\\_config\\_t::whichPcs](#)
- (4) bool [dspi\\_command\\_data\\_config\\_t::isEndOfQueue](#)
- (5) bool [dspi\\_command\\_data\\_config\\_t::clearTransferCount](#)

### 11.2.3.2 struct dspi\_master\_ctar\_config\_t

#### Data Fields

- uint32\_t [baudRate](#)  
*Baud Rate for DSPI.*
- uint32\_t [bitsPerFrame](#)  
*Bits per frame, minimum 4, maximum 16.*
- [dspi\\_clock\\_polarity\\_t](#) [cpol](#)  
*Clock polarity.*
- [dspi\\_clock\\_phase\\_t](#) [cpha](#)  
*Clock phase.*
- [dspi\\_shift\\_direction\\_t](#) [direction](#)  
*MSB or LSB data shift direction.*
- uint32\_t [pcsToSckDelayInNanoSec](#)  
*PCS to SCK delay time in nanoseconds; setting to 0 sets the minimum delay.*
- uint32\_t [lastSckToPcsDelayInNanoSec](#)  
*The last SCK to PCS delay time in nanoseconds; setting to 0 sets the minimum delay.*
- uint32\_t [betweenTransferDelayInNanoSec](#)  
*After the SCK delay time in nanoseconds; setting to 0 sets the minimum delay.*

## Field Documentation

- (1) `uint32_t dspi_master_ctar_config_t::baudRate`
- (2) `uint32_t dspi_master_ctar_config_t::bitsPerFrame`
- (3) `dspi_clock_polarity_t dspi_master_ctar_config_t::cpol`
- (4) `dspi_clock_phase_t dspi_master_ctar_config_t::cpha`
- (5) `dspi_shift_direction_t dspi_master_ctar_config_t::direction`
- (6) `uint32_t dspi_master_ctar_config_t::pcsToSckDelayInNanoSec`

It also sets the boundary value if out of range.

- (7) `uint32_t dspi_master_ctar_config_t::lastSckToPcsDelayInNanoSec`

It also sets the boundary value if out of range.

- (8) `uint32_t dspi_master_ctar_config_t::betweenTransferDelayInNanoSec`

It also sets the boundary value if out of range.

### 11.2.3.3 struct `dspi_master_config_t`

#### Data Fields

- `dspi_ctar_selection_t whichCtar`  
*The desired CTAR to use.*
- `dspi_master_ctar_config_t ctarConfig`  
*Set the ctarConfig to the desired CTAR.*
- `dspi_which_pcs_t whichPcs`  
*The desired Peripheral Chip Select (pcs).*
- `dspi_pcs_polarity_config_t pcsActiveHighOrLow`  
*The desired PCS active high or low.*
- `bool enableContinuousSCK`  
*CONT\_SCKE, continuous SCK enable.*
- `bool enableRxFifoOverWrite`  
*ROOE, receive FIFO overflow overwrite enable.*
- `bool enableModifiedTimingFormat`  
*Enables a modified transfer format to be used if true.*
- `dspi_master_sample_point_t samplePoint`  
*Controls when the module master samples SIN in the Modified Transfer Format.*

## Field Documentation

- (1) `dspi_ctar_selection_t dspi_master_config_t::whichCtar`

- (2) **dspi\_master\_ctar\_config\_t dspi\_master\_config\_t::ctarConfig**
- (3) **dspi\_which\_pcs\_t dspi\_master\_config\_t::whichPcs**
- (4) **dspi\_pcs\_polarity\_config\_t dspi\_master\_config\_t::pcsActiveHighOrLow**
- (5) **bool dspi\_master\_config\_t::enableContinuousSCK**

Note that the continuous SCK is only supported for CPHA = 1.

- (6) **bool dspi\_master\_config\_t::enableRx\_fifo\_overWrite**

If ROOE = 0, the incoming data is ignored and the data from the transfer that generated the overflow is also ignored. If ROOE = 1, the incoming data is shifted to the shift register.

- (7) **bool dspi\_master\_config\_t::enableModifiedTimingFormat**
- (8) **dspi\_master\_sample\_point\_t dspi\_master\_config\_t::samplePoint**

It's valid only when CPHA=0.

#### 11.2.3.4 struct dspi\_slave\_ctar\_config\_t

##### Data Fields

- **uint32\_t bitsPerFrame**  
*Bits per frame, minimum 4, maximum 16.*
- **dspi\_clock\_polarity\_t cpol**  
*Clock polarity.*
- **dspi\_clock\_phase\_t cpha**  
*Clock phase.*

##### Field Documentation

- (1) **uint32\_t dspi\_slave\_ctar\_config\_t::bitsPerFrame**
- (2) **dspi\_clock\_polarity\_t dspi\_slave\_ctar\_config\_t::cpol**
- (3) **dspi\_clock\_phase\_t dspi\_slave\_ctar\_config\_t::cpha**

Slave only supports MSB and does not support LSB.

#### 11.2.3.5 struct dspi\_slave\_config\_t

##### Data Fields

- **dspi\_ctar\_selection\_t whichCtar**  
*The desired CTAR to use.*
- **dspi\_slave\_ctar\_config\_t ctarConfig**

- Set the `ctarConfig` to the desired CTAR.
- bool `enableContinuousSCK`  
*CONT\_SCKE, continuous SCK enable.*
- bool `enableRx_fifo_OverWrite`  
*ROOE, receive FIFO overflow overwrite enable.*
- bool `enableModifiedTimingFormat`  
*Enables a modified transfer format to be used if true.*
- `dspi_master_sample_point_t samplePoint`  
*Controls when the module master samples SIN in the Modified Transfer Format.*

### Field Documentation

- (1) `dspi_ctar_selection_t dspi_slave_config_t::whichCtar`
- (2) `dspi_slave_ctar_config_t dspi_slave_config_t::ctarConfig`
- (3) `bool dspi_slave_config_t::enableContinuousSCK`

Note that the continuous SCK is only supported for CPHA = 1.

- (4) `bool dspi_slave_config_t::enableRx_fifo_OverWrite`

If ROOE = 0, the incoming data is ignored and the data from the transfer that generated the overflow is also ignored. If ROOE = 1, the incoming data is shifted to the shift register.

- (5) `bool dspi_slave_config_t::enableModifiedTimingFormat`
- (6) `dspi_master_sample_point_t dspi_slave_config_t::samplePoint`

It's valid only when CPHA=0.

### 11.2.3.6 struct `dspi_transfer_t`

#### Data Fields

- `uint8_t * txData`  
*Send buffer.*
- `uint8_t * rxData`  
*Receive buffer.*
- `volatile size_t dataSize`  
*Transfer bytes.*
- `uint32_t configFlags`  
*Transfer transfer configuration flags.*

### Field Documentation

- (1) `uint8_t* dspi_transfer_t::txData`
- (2) `uint8_t* dspi_transfer_t::rxData`

(3) **volatile size\_t dspi\_transfer\_t::dataSize**

(4) **uint32\_t dspi\_transfer\_t::configFlags**

Set from [\\_dsPIC33F\\_DSPI\\_Transfer\\_Config\\_Flag\\_for\\_Master](#) if the transfer is used for master or [\\_dsPIC33F\\_DSPI\\_Transfer\\_Config\\_Flag\\_for\\_Slave](#) enumeration if the transfer is used for slave.

### 11.2.3.7 struct dspi\_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; set from [\\_dsPIC33F\\_DSPI\\_Transfer\\_Config\\_Flag\\_for\\_Master](#).*
- **bool isPcsAssertInTransfer**  
*If Pcs pin keep assert between transmit and receive.*
- **bool isTransmitFirst**  
*True for transmit first and false for receive first.*

#### Field Documentation

(1) **uint32\_t dspi\_half\_duplex\_transfer\_t::configFlags**

(2) **bool dspi\_half\_duplex\_transfer\_t::isPcsAssertInTransfer**

true for assert and false for de-assert.

(3) **bool dspi\_half\_duplex\_transfer\_t::isTransmitFirst**

### 11.2.3.8 struct \_dsPIC33F\_DSPI\_Master\_Handle

Forward declaration of the [\\_dsPIC33F\\_DSPI\\_Master\\_Handle](#) typedefs.

The master handle.

#### Data Fields

- **uint32\_t bitsPerFrame**  
*The desired number of bits per frame.*
- **volatile uint32\_t command**  
*The desired data command.*
- **volatile uint32\_t lastCommand**

- **uint8\_t fifoSize**  
*FIFO dataSize.*
- **volatile bool isPcsActiveAfterTransfer**  
*Indicates whether the PCS signal is active after the last frame transfer.*
- **volatile bool isThereExtraByte**  
*Indicates whether there are extra bytes.*
- **uint8\_t \*volatile txData**  
*Send buffer.*
- **uint8\_t \*volatile rxData**  
*Receive buffer.*
- **volatile size\_t remainingSendByteCount**  
*A number of bytes remaining to send.*
- **volatile size\_t remainingReceiveByteCount**  
*A number of bytes remaining to receive.*
- **size\_t totalByteCount**  
*A number of transfer bytes.*
- **volatile uint8\_t state**  
*DSPI transfer state, see [\\_dspi\\_transfer\\_state](#).*
- **dspi\_master\_transfer\_callback\_t callback**  
*Completion callback.*
- **void \*userData**  
*Callback user data.*

## Field Documentation

- (1) **uint32\_t dspi\_master\_handle\_t::bitsPerFrame**
- (2) **volatile uint32\_t dspi\_master\_handle\_t::command**
- (3) **volatile uint32\_t dspi\_master\_handle\_t::lastCommand**
- (4) **uint8\_t dspi\_master\_handle\_t::fifoSize**
- (5) **volatile bool dspi\_master\_handle\_t::isPcsActiveAfterTransfer**
- (6) **volatile bool dspi\_master\_handle\_t::isThereExtraByte**
- (7) **uint8\_t\* volatile dspi\_master\_handle\_t::txData**
- (8) **uint8\_t\* volatile dspi\_master\_handle\_t::rxData**
- (9) **volatile size\_t dspi\_master\_handle\_t::remainingSendByteCount**
- (10) **volatile size\_t dspi\_master\_handle\_t::remainingReceiveByteCount**
- (11) **volatile uint8\_t dspi\_master\_handle\_t::state**
- (12) **dspi\_master\_transfer\_callback\_t dspi\_master\_handle\_t::callback**
- (13) **void\* dspi\_master\_handle\_t::userData**

### 11.2.3.9 struct \_dspi\_slave\_handle

Forward declaration of the `_dspi_slave_handle` typedefs.

The slave handle.

#### Data Fields

- `uint32_t bitsPerFrame`  
*The desired number of bits per frame.*
- `volatile bool isThereExtraByte`  
*Indicates whether there are extra bytes.*
- `uint8_t *volatile txData`  
*Send buffer.*
- `uint8_t *volatile rxData`  
*Receive buffer.*
- `volatile size_t remainingSendByteCount`  
*A number of bytes remaining to send.*
- `volatile size_t remainingReceiveByteCount`  
*A number of bytes remaining to receive.*
- `size_t totalByteCount`  
*A number of transfer bytes.*
- `volatile uint8_t state`  
*DSPI transfer state.*
- `volatile uint32_t errorCount`  
*Error count for slave transfer.*
- `dspi_slave_transfer_callback_t callback`  
*Completion callback.*
- `void *userData`  
*Callback user data.*

#### Field Documentation

- (1) `uint32_t dspi_slave_handle_t::bitsPerFrame`
- (2) `volatile bool dspi_slave_handle_t::isThereExtraByte`
- (3) `uint8_t* volatile dspi_slave_handle_t::txData`
- (4) `uint8_t* volatile dspi_slave_handle_t::rxData`
- (5) `volatile size_t dspi_slave_handle_t::remainingSendByteCount`
- (6) `volatile size_t dspi_slave_handle_t::remainingReceiveByteCount`
- (7) `volatile uint8_t dspi_slave_handle_t::state`
- (8) `volatile uint32_t dspi_slave_handle_t::errorCount`
- (9) `dspi_slave_transfer_callback_t dspi_slave_handle_t::callback`

(10) `void* dspi_slave_handle_t::userData`

#### 11.2.4 Macro Definition Documentation

**11.2.4.1 `#define FSL_DSPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 4))`**

**11.2.4.2 `#define DSPI_DUMMY_DATA (0x00U)`**

Dummy data used for Tx if there is no txData.

**11.2.4.3 `#define DSPI_MASTER_CTAR_SHIFT (0U)`**

**11.2.4.4 `#define DSPI_MASTER_CTAR_MASK (0x0FU)`**

**11.2.4.5 `#define DSPI_MASTER_PCS_SHIFT (4U)`**

**11.2.4.6 `#define DSPI_MASTER_PCS_MASK (0xF0U)`**

**11.2.4.7 `#define DSPI_SLAVE_CTAR_SHIFT (0U)`**

**11.2.4.8 `#define DSPI_SLAVE_CTAR_MASK (0x07U)`**

#### 11.2.5 Typedef Documentation

**11.2.5.1 `typedef void(* dspi_master_transfer_callback_t)(SPI_Type *base, dspi_master_handle_t *handle, status_t status, void *userData)`**

Parameters

|                 |                                                                  |
|-----------------|------------------------------------------------------------------|
| <i>base</i>     | DSPI peripheral address.                                         |
| <i>handle</i>   | Pointer to the handle for the DSPI master.                       |
| <i>status</i>   | Success or error code describing whether the transfer completed. |
| <i>userData</i> | Arbitrary pointer-dataSized value passed from the application.   |

**11.2.5.2 `typedef void(* dspi_slave_transfer_callback_t)(SPI_Type *base, dspi_slave_handle_t *handle, status_t status, void *userData)`**

Parameters

|                 |                                                                  |
|-----------------|------------------------------------------------------------------|
| <i>base</i>     | DSPI peripheral address.                                         |
| <i>handle</i>   | Pointer to the handle for the DSPI slave.                        |
| <i>status</i>   | Success or error code describing whether the transfer completed. |
| <i>userData</i> | Arbitrary pointer-dataSized value passed from the application.   |

## 11.2.6 Enumeration Type Documentation

### 11.2.6.1 anonymous enum

Enumerator

- kStatus\_DSPI\_Busy* DSPI transfer is busy.
- kStatus\_DSPI\_Error* DSPI driver error.
- kStatus\_DSPI\_Idle* DSPI is idle.
- kStatus\_DSPI\_OutOfRange* DSPI transfer out of range.

### 11.2.6.2 enum \_dspi\_flags

Enumerator

- kDSPI\_TxCompleteFlag* Transfer Complete Flag.
- kDSPI\_EndOfQueueFlag* End of Queue Flag.
- kDSPI\_TxFifoUnderflowFlag* Transmit FIFO Underflow Flag.
- kDSPI\_TxFifoFillRequestFlag* Transmit FIFO Fill Flag.
- kDSPI\_RxFifoOverflowFlag* Receive FIFO Overflow Flag.
- kDSPI\_RxFifoDrainRequestFlag* Receive FIFO Drain Flag.
- kDSPI\_TxAndRxStatusFlag* The module is in Stopped/Running state.
- kDSPI\_AllStatusFlag* All statuses above.

### 11.2.6.3 enum \_dspi\_interrupt\_enable

Enumerator

- kDSPI\_TxCompleteInterruptEnable* TCF interrupt enable.
- kDSPI\_EndOfQueueInterruptEnable* EOQF interrupt enable.
- kDSPI\_TxFifoUnderflowInterruptEnable* TFUF interrupt enable.
- kDSPI\_TxFifoFillRequestInterruptEnable* TFFF interrupt enable, DMA disable.
- kDSPI\_RxFifoOverflowInterruptEnable* RFOF interrupt enable.
- kDSPI\_RxFifoDrainRequestInterruptEnable* RFDF interrupt enable, DMA disable.
- kDSPI\_AllInterruptEnable* All above interrupts enable.

#### 11.2.6.4 enum \_dspi\_dma\_enable

Enumerator

*kDSPI\_TxDmaEnable* TFFF flag generates DMA requests. No Tx interrupt request.

*kDSPI\_RxDmaEnable* RFDF flag generates DMA requests. No Rx interrupt request.

#### 11.2.6.5 enum dspi\_master\_slave\_mode\_t

Enumerator

*kDSPI\_Master* DSPI peripheral operates in master mode.

*kDSPI\_Slave* DSPI peripheral operates in slave mode.

#### 11.2.6.6 enum dspi\_master\_sample\_point\_t

This field is valid only when the CPHA bit in the CTAR register is 0.

Enumerator

*kDSPI\_SckToSin0Clock* 0 system clocks between SCK edge and SIN sample.

*kDSPI\_SckToSin1Clock* 1 system clock between SCK edge and SIN sample.

*kDSPI\_SckToSin2Clock* 2 system clocks between SCK edge and SIN sample.

#### 11.2.6.7 enum dspi\_which\_pcs\_t

Enumerator

*kDSPI\_Pcs0* Pcs[0].

*kDSPI\_Pcs1* Pcs[1].

*kDSPI\_Pcs2* Pcs[2].

*kDSPI\_Pcs3* Pcs[3].

*kDSPI\_Pcs4* Pcs[4].

*kDSPI\_Pcs5* Pcs[5].

#### 11.2.6.8 enum dspi\_pcs\_polarity\_config\_t

Enumerator

*kDSPI\_PcsActiveHigh* Pcs Active High (idles low).

*kDSPI\_PcsActiveLow* Pcs Active Low (idles high).

### 11.2.6.9 enum \_dspi\_pcs\_polarity

Enumerator

- kDSPI\_Pcs0ActiveLow* Pcs0 Active Low (idles high).
- kDSPI\_Pcs1ActiveLow* Pcs1 Active Low (idles high).
- kDSPI\_Pcs2ActiveLow* Pcs2 Active Low (idles high).
- kDSPI\_Pcs3ActiveLow* Pcs3 Active Low (idles high).
- kDSPI\_Pcs4ActiveLow* Pcs4 Active Low (idles high).
- kDSPI\_Pcs5ActiveLow* Pcs5 Active Low (idles high).
- kDSPI\_PcsAllActiveLow* Pcs0 to Pcs5 Active Low (idles high).

### 11.2.6.10 enum dspi\_clock\_polarity\_t

Enumerator

- kDSPI\_ClockPolarityActiveHigh* CPOL=0. Active-high DSPI clock (idles low).
- kDSPI\_ClockPolarityActiveLow* CPOL=1. Active-low DSPI clock (idles high).

### 11.2.6.11 enum dspi\_clock\_phase\_t

Enumerator

- kDSPI\_ClockPhaseFirstEdge* CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.
- kDSPI\_ClockPhaseSecondEdge* CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

### 11.2.6.12 enum dspi\_shift\_direction\_t

Enumerator

- kDSPI\_MsbFirst* Data transfers start with most significant bit.
- kDSPI\_LsbFirst* Data transfers start with least significant bit. Shifting out of LSB is not supported for slave

### 11.2.6.13 enum dspi\_delay\_type\_t

Enumerator

- kDSPI\_PcsToSck* Pcs-to-SCK delay.
- kDSPI\_LastSckToPcs* The last SCK edge to Pcs delay.
- kDSPI\_BetweenTransfer* Delay between transfers.

### 11.2.6.14 enum dspi\_ctar\_selection\_t

Enumerator

- kDSPI\_Ctar0*** CTAR0 selection option for master or slave mode; note that CTAR0 and CTAR0\_S-LAVE are the same register address.
- kDSPI\_Ctar1*** CTAR1 selection option for master mode only.
- kDSPI\_Ctar2*** CTAR2 selection option for master mode only; note that some devices do not support CTAR2.
- kDSPI\_Ctar3*** CTAR3 selection option for master mode only; note that some devices do not support CTAR3.
- kDSPI\_Ctar4*** CTAR4 selection option for master mode only; note that some devices do not support CTAR4.
- kDSPI\_Ctar5*** CTAR5 selection option for master mode only; note that some devices do not support CTAR5.
- kDSPI\_Ctar6*** CTAR6 selection option for master mode only; note that some devices do not support CTAR6.
- kDSPI\_Ctar7*** CTAR7 selection option for master mode only; note that some devices do not support CTAR7.

### 11.2.6.15 enum \_dspi\_transfer\_config\_flag\_for\_master

Enumerator

- kDSPI\_MasterCtar0*** DSPI master transfer use CTAR0 setting.
- kDSPI\_MasterCtar1*** DSPI master transfer use CTAR1 setting.
- kDSPI\_MasterCtar2*** DSPI master transfer use CTAR2 setting.
- kDSPI\_MasterCtar3*** DSPI master transfer use CTAR3 setting.
- kDSPI\_MasterCtar4*** DSPI master transfer use CTAR4 setting.
- kDSPI\_MasterCtar5*** DSPI master transfer use CTAR5 setting.
- kDSPI\_MasterCtar6*** DSPI master transfer use CTAR6 setting.
- kDSPI\_MasterCtar7*** DSPI master transfer use CTAR7 setting.
- kDSPI\_MasterPcs0*** DSPI master transfer use PCS0 signal.
- kDSPI\_MasterPcs1*** DSPI master transfer use PCS1 signal.
- kDSPI\_MasterPcs2*** DSPI master transfer use PCS2 signal.
- kDSPI\_MasterPcs3*** DSPI master transfer use PCS3 signal.
- kDSPI\_MasterPcs4*** DSPI master transfer use PCS4 signal.
- kDSPI\_MasterPcs5*** DSPI master transfer use PCS5 signal.
- kDSPI\_MasterPcsContinuous*** Indicates whether the PCS signal is continuous.
- kDSPI\_MasterActiveAfterTransfer*** Indicates whether the PCS signal is active after the last frame transfer.

### 11.2.6.16 enum \_dsPIC\_transfer\_config\_flag\_for\_slave

Enumerator

**kDSPI\_SlaveCtar0** DSPI slave transfer use CTAR0 setting. DSPI slave can only use PCS0.

### 11.2.6.17 enum \_dsPIC\_transfer\_state

Enumerator

**kDSPI\_Idle** Nothing in the transmitter/receiver.

**kDSPI\_Busy** Transfer queue is not finished.

**kDSPI\_Error** Transfer error.

## 11.2.7 Function Documentation

### 11.2.7.1 void DSPI\_MasterInit ( SPI\_Type \* base, const dsPIC\_master\_config\_t \* masterConfig, uint32\_t srcClock\_Hz )

This function initializes the DSPI master configuration. This is an example use case.

```
* dsPIC_master_config_t masterConfig;
* masterConfig.whichCtar = kDSPI_Ctar0;
* masterConfig.ctarConfig.baudRate = 500000000U;
* masterConfig.ctarConfig.bitsPerFrame = 8;
* masterConfig.ctarConfig.cpol =
* kDSPI_ClockPolarityActiveHigh;
* masterConfig.ctarConfig.cpha =
* kDSPI_ClockPhaseFirstEdge;
* masterConfig.ctarConfig.direction =
* kDSPI_MsbFirst;
* masterConfig.ctarConfig.pcsToSckDelayInNanoSec =
* 10000000000U /
* masterConfig.ctarConfig.baudRate ;
* masterConfig.ctarConfig.lastSckToPcsDelayInNanoSec =
* 10000000000U /
* masterConfig.ctarConfig.baudRate ;
* masterConfig.ctarConfig.betweenTransferDelayInNanoSec =
* 10000000000U / masterConfig.ctarConfig.baudRate ;
* masterConfig.whichPcs = kDSPI_Pcs0;
* masterConfig.pcsActiveHighOrLow =
* kDSPI_PcsActiveLow;
* masterConfig.enableContinuousSCK =
* false;
* masterConfig.enableRx_fifoOverWrite =
* false;
* masterConfig.enableModifiedTimingFormat =
* false;
* masterConfig.samplePoint =
* kDSPI_SckToSinc0Clock;
* DSPI_MasterInit (base, &masterConfig, srcClock_Hz);
*
```

## Parameters

|                     |                                                                 |
|---------------------|-----------------------------------------------------------------|
| <i>base</i>         | DSPI peripheral address.                                        |
| <i>masterConfig</i> | Pointer to the structure <a href="#">dspi_master_config_t</a> . |
| <i>srcClock_Hz</i>  | Module source input clock in Hertz.                             |

**11.2.7.2 void DSPI\_MasterGetDefaultConfig ( [dspi\\_master\\_config\\_t](#) \* *masterConfig* )**

The purpose of this API is to get the configuration structure initialized for the [DSPI\\_MasterInit\(\)](#). Users may use the initialized structure unchanged in the [DSPI\\_MasterInit\(\)](#) or modify the structure before calling the [DSPI\\_MasterInit\(\)](#). Example:

```
* dspi_master_config_t masterConfig;
* DSPI_MasterGetDefaultConfig(&masterConfig);
*
```

## Parameters

|                     |                                                           |
|---------------------|-----------------------------------------------------------|
| <i>masterConfig</i> | pointer to <a href="#">dspi_master_config_t</a> structure |
|---------------------|-----------------------------------------------------------|

**11.2.7.3 void DSPI\_SlaveInit ( [SPI\\_Type](#) \* *base*, [const dspi\\_slave\\_config\\_t](#) \* *slaveConfig* )**

This function initializes the DSPI slave configuration. This is an example use case.

```
* dspi_slave_config_t slaveConfig;
* slaveConfig->whichCtar = kDSPI_Ctar0;
* slaveConfig->cstarConfig.bitsPerFrame = 8;
* slaveConfig->cstarConfig.cpol =
 kDSPI_ClockPolarityActiveHigh;
* slaveConfig->cstarConfig.cpha =
 kDSPI_ClockPhaseFirstEdge;
* slaveConfig->enableContinuousSCK = false;
* slaveConfig->enableRx_fifoOverWrite = false;
* slaveConfig->enableModifiedTimingFormat = false;
* slaveConfig->samplePoint = kDSPI_SckToSin0Clock;
* DSPI_SlaveInit(base, &slaveConfig);
*
```

## Parameters

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

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>slaveConfig</i> | Pointer to the structure <a href="#">dspi_slave_config_t</a> . |
|--------------------|----------------------------------------------------------------|

#### 11.2.7.4 void DSPI\_SlaveGetDefaultConfig ( [dspi\\_slave\\_config\\_t](#) \* *slaveConfig* )

The purpose of this API is to get the configuration structure initialized for the [DSPI\\_SlaveInit\(\)](#). Users may use the initialized structure unchanged in the [DSPI\\_SlaveInit\(\)](#) or modify the structure before calling the [DSPI\\_SlaveInit\(\)](#). This is an example.

```
* dspi_slave_config_t slaveConfig;
* DSPI_SlaveGetDefaultConfig(&slaveConfig);
*
```

Parameters

|                    |                                                               |
|--------------------|---------------------------------------------------------------|
| <i>slaveConfig</i> | Pointer to the <a href="#">dspi_slave_config_t</a> structure. |
|--------------------|---------------------------------------------------------------|

#### 11.2.7.5 void DSPI\_Deinit ( [SPI\\_Type](#) \* *base* )

Call this API to disable the DSPI clock.

Parameters

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

#### 11.2.7.6 static void DSPI\_Enable ( [SPI\\_Type](#) \* *base*, [bool](#) *enable* ) [inline], [static]

Parameters

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>base</i>   | DSPI peripheral address.                             |
| <i>enable</i> | Pass true to enable module, false to disable module. |

#### 11.2.7.7 static [uint32\\_t](#) DSPI\_GetStatusFlags ( [SPI\\_Type](#) \* *base* ) [inline], [static]

## Parameters

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

## Returns

DSPI status (in SR register).

### 11.2.7.8 static void DSPI\_ClearStatusFlags ( SPI\_Type \* *base*, uint32\_t *statusFlags* ) [inline], [static]

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status bit to clear. The list of status bits is defined in the **dspi\_status\_and\_interrupt\_request\_t**. The function uses these bit positions in its algorithm to clear the desired flag state. This is an example.

```
* DSPI_ClearStatusFlags(base, kDSPI_TxCompleteFlag |
 kDSPI_EndOfQueueFlag);
*
```

## Parameters

|                    |                                                        |
|--------------------|--------------------------------------------------------|
| <i>base</i>        | DSPI peripheral address.                               |
| <i>statusFlags</i> | The status flag used from the type <b>dspi_flags</b> . |

< The status flags are cleared by writing 1 (w1c).

### 11.2.7.9 void DSPI\_EnableInterrupts ( SPI\_Type \* *base*, uint32\_t *mask* )

This function configures various interrupt masks of the DSPI. The parameters are a base and an interrupt mask.

## Note

For Tx Fill and Rx FIFO drain requests, enable the interrupt request and disable the DMA request.  
Do not use this API(write to RSER register) while DSPI is in running state.

```
* DSPI_EnableInterrupts(base,
 kDSPI_TxCompleteInterruptEnable |
 kDSPI_EndOfQueueInterruptEnable);
*
```

Parameters

|             |                                                                           |
|-------------|---------------------------------------------------------------------------|
| <i>base</i> | DSPI peripheral address.                                                  |
| <i>mask</i> | The interrupt mask; use the enum <a href="#">_dspi_interrupt_enable</a> . |

#### 11.2.7.10 static void DSPI\_DisableInterrupts ( SPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

```
* DSPI_DisableInterrupts(base,
 kDSPI_TxCompleteInterruptEnable |
 kDSPI_EndOfQueueInterruptEnable);
*
```

Parameters

|             |                                                                           |
|-------------|---------------------------------------------------------------------------|
| <i>base</i> | DSPI peripheral address.                                                  |
| <i>mask</i> | The interrupt mask; use the enum <a href="#">_dspi_interrupt_enable</a> . |

#### 11.2.7.11 static void DSPI\_EnableDMA ( SPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function configures the Rx and Tx DMA mask of the DSPI. The parameters are a base and a DMA mask.

```
* DSPI_EnableDMA(base, kDSPI_TxDmaEnable |
 kDSPI_RxDmaEnable);
*
```

Parameters

|             |                                                                     |
|-------------|---------------------------------------------------------------------|
| <i>base</i> | DSPI peripheral address.                                            |
| <i>mask</i> | The interrupt mask; use the enum <a href="#">_dspi_dma_enable</a> . |

#### 11.2.7.12 static void DSPI\_DisableDMA ( SPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function configures the Rx and Tx DMA mask of the DSPI. The parameters are a base and a DMA mask.

```
* SPI_DisableDMA(base, kDSPI_TxDmaEnable | kDSPI_RxDmaEnable);
*
```

Parameters

|             |                                                                     |
|-------------|---------------------------------------------------------------------|
| <i>base</i> | DSPI peripheral address.                                            |
| <i>mask</i> | The interrupt mask; use the enum <a href="#">_dspi_dma_enable</a> . |

#### **11.2.7.13 static uint32\_t DSPI\_MasterGetTxRegisterAddress ( SPI\_Type \* *base* ) [inline], [static]**

This function gets the DSPI master PUSHR data register address because this value is needed for the DMA operation.

Parameters

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

Returns

The DSPI master PUSHR data register address.

#### **11.2.7.14 static uint32\_t DSPI\_SlaveGetTxRegisterAddress ( SPI\_Type \* *base* ) [inline], [static]**

This function gets the DSPI slave PUSHR data register address as this value is needed for the DMA operation.

Parameters

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

Returns

The DSPI slave PUSHR data register address.

#### **11.2.7.15 static uint32\_t DSPI\_GetRxRegisterAddress ( SPI\_Type \* *base* ) [inline], [static]**

This function gets the DSPI POPR data register address as this value is needed for the DMA operation.

Parameters

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

Returns

The DSPI POPR data register address.

#### 11.2.7.16 `uint32_t DSPI_GetInstance ( SPI_Type * base )`

Parameters

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

#### 11.2.7.17 `static void DSPI_SetMasterSlaveMode ( SPI_Type * base, dspi_master_slave_mode_t mode ) [inline], [static]`

Parameters

|             |                                                                                |
|-------------|--------------------------------------------------------------------------------|
| <i>base</i> | DSPI peripheral address.                                                       |
| <i>mode</i> | Mode setting (master or slave) of type <code>dspi_master_slave_mode_t</code> . |

#### 11.2.7.18 `static bool DSPI_IsMaster ( SPI_Type * base ) [inline], [static]`

Parameters

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

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

#### 11.2.7.19 `static void DSPI_StartTransfer ( SPI_Type * base ) [inline], [static]`

This function sets the module to start data transfer in either master or slave mode.

Parameters

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

#### 11.2.7.20 static void DSPI\_StopTransfer ( SPI\_Type \* *base* ) [inline], [static]

This function stops data transfers in either master or slave modes.

Parameters

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

#### 11.2.7.21 static void DSPI\_SetFifoEnable ( SPI\_Type \* *base*, bool *enableTxFifo*, bool *enableRxFifo* ) [inline], [static]

This function allows the caller to disable/enable the Tx and Rx FIFOs independently.

Note

To disable, pass in a logic 0 (false) for the particular FIFO configuration. To enable, pass in a logic 1 (true).

Parameters

|                     |                                                                     |
|---------------------|---------------------------------------------------------------------|
| <i>base</i>         | DSPI peripheral address.                                            |
| <i>enableTxFifo</i> | Disables (false) the TX FIFO; Otherwise, enables (true) the TX FIFO |
| <i>enableRxFifo</i> | Disables (false) the RX FIFO; Otherwise, enables (true) the RX FIFO |

#### 11.2.7.22 static void DSPI\_FlushFifo ( SPI\_Type \* *base*, bool *flushTxFifo*, bool *flushRxFifo* ) [inline], [static]

Parameters

|                    |                                                                           |
|--------------------|---------------------------------------------------------------------------|
| <i>base</i>        | DSPI peripheral address.                                                  |
| <i>flushTxFifo</i> | Flushes (true) the Tx FIFO; Otherwise, does not flush (false) the Tx FIFO |

|                    |                                                                           |
|--------------------|---------------------------------------------------------------------------|
| <i>flushRxFifo</i> | Flushes (true) the Rx FIFO; Otherwise, does not flush (false) the Rx FIFO |
|--------------------|---------------------------------------------------------------------------|

### 11.2.7.23 static void DSPI\_SetAllPcsPolarity ( SPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

For example, PCS0 and PCS1 are set to active low and other PCS is set to active high. Note that the number of PCSs is specific to the device.

```
* DSPI_SetAllPcsPolarity(base, kDSPI_Pcs0ActiveLow |
 kDSPI_Pcs1ActiveLow);
```

Parameters

|             |                                                                          |
|-------------|--------------------------------------------------------------------------|
| <i>base</i> | DSPI peripheral address.                                                 |
| <i>mask</i> | The PCS polarity mask; use the enum <a href="#">_dspi_pcs_polarity</a> . |

### 11.2.7.24 uint32\_t DSPI\_MasterSetBaudRate ( SPI\_Type \* *base*, dsPICtar\_selection\_t whichCtar, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz )

This function takes in the desired baudRate\_Bps (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate, and returns the calculated baud rate in bits-per-second. It requires that the caller also provide the frequency of the module source clock (in Hertz).

Parameters

|                     |                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | DSPI peripheral address.                                                                                   |
| <i>whichCtar</i>    | The desired Clock and Transfer Attributes Register (CTAR) of the type <a href="#">dsPICtar_selection_t</a> |
| <i>baudRate_Bps</i> | The desired baud rate in bits per second                                                                   |
| <i>srcClock_Hz</i>  | Module source input clock in Hertz                                                                         |

Returns

The actual calculated baud rate

### 11.2.7.25 void DSPI\_MasterSetDelayScaler ( SPI\_Type \* *base*, dsPICtar\_selection\_t whichCtar, uint32\_t prescaler, uint32\_t scaler, dsPICdelay\_type\_t whichDelay )

This function configures the PCS to SCK delay pre-scalar (PcsSCK) and scalar (CSSCK), after SCK delay pre-scalar (PASC) and scalar (ASC), and the delay after transfer pre-scalar (PDT) and scalar (DT).

These delay names are available in the type [dspi\\_delay\\_type\\_t](#).

The user passes the delay to the configuration along with the prescaler and scaler value. This allows the user to directly set the prescaler/scaler values if pre-calculated or to manually increment either value.

Parameters

|                   |                                                                                                           |
|-------------------|-----------------------------------------------------------------------------------------------------------|
| <i>base</i>       | DSPI peripheral address.                                                                                  |
| <i>whichCtar</i>  | The desired Clock and Transfer Attributes Register (CTAR) of type <a href="#">dspi_ctar_selection_t</a> . |
| <i>prescaler</i>  | The prescaler delay value (can be an integer 0, 1, 2, or 3).                                              |
| <i>scaler</i>     | The scaler delay value (can be any integer between 0 to 15).                                              |
| <i>whichDelay</i> | The desired delay to configure; must be of type <a href="#">dspi_delay_type_t</a>                         |

#### **11.2.7.26 uint32\_t DSPI\_MasterSetDelayTimes ( SPI\_Type \* *base*, [dspi\\_ctar\\_selection\\_t](#) *whichCtar*, [dspi\\_delay\\_type\\_t](#) *whichDelay*, uint32\_t *srcClock\_Hz*, uint32\_t *delayTimeInNanoSec* )**

This function calculates the values for the following. PCS to SCK delay pre-scalar (PCSSCK) and scalar (CSSCK), or After SCK delay pre-scalar (PASC) and scalar (ASC), or Delay after transfer pre-scalar (PDT) and scalar (DT).

These delay names are available in the type [dspi\\_delay\\_type\\_t](#).

The user passes which delay to configure along with the desired delay value in nanoseconds. The function calculates the values needed for the prescaler and scaler. Note that returning the calculated delay as an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. The higher-level peripheral driver alerts the user of an out of range delay input.

Parameters

|                    |                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------|
| <i>base</i>        | DSPI peripheral address.                                                                                  |
| <i>whichCtar</i>   | The desired Clock and Transfer Attributes Register (CTAR) of type <a href="#">dspi_ctar_selection_t</a> . |
| <i>whichDelay</i>  | The desired delay to configure, must be of type <a href="#">dspi_delay_type_t</a>                         |
| <i>srcClock_Hz</i> | Module source input clock in Hertz                                                                        |

|                            |                                         |
|----------------------------|-----------------------------------------|
| <i>delayTimeIn-NanoSec</i> | The desired delay value in nanoseconds. |
|----------------------------|-----------------------------------------|

Returns

The actual calculated delay value.

#### 11.2.7.27 static void DSPI\_MasterWriteData ( SPI\_Type \* *base*, dspi-command\_data\_config\_t \* *command*, uint16\_t *data* ) [inline], [static]

In master mode, the 16-bit data is appended to the 16-bit command info. The command portion provides characteristics of the data, such as the optional continuous chip select operation between transfers, the desired Clock and Transfer Attributes register to use for the associated SPI frame, the desired PCS signal to use for the data transfer, whether the current transfer is the last in the queue, and whether to clear the transfer count (normally needed when sending the first frame of a data packet). This is an example.

```
* dspi_command_data_config_t commandConfig;
* commandConfig.isPcsContinuous = true;
* commandConfig.whichCtar = kDSPICTar0;
* commandConfig.whichPcs = kDSPIPcs0;
* commandConfig.clearTransferCount = false;
* commandConfig.isEndOfQueue = false;
* DSPI_MasterWriteData(base, &commandConfig, dataWord);
```

Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>base</i>    | DSPI peripheral address.          |
| <i>command</i> | Pointer to the command structure. |
| <i>data</i>    | The data word to be sent.         |

#### 11.2.7.28 void DSPI\_GetDefaultDataCommandConfig ( dspi\_command\_data\_config\_t \* *command* )

The purpose of this API is to get the configuration structure initialized for use in the **DSPI\_MasterWrite\_xx()**. Users may use the initialized structure unchanged in the **DSPI\_MasterWrite\_xx()** or modify the structure before calling the **DSPI\_MasterWrite\_xx()**. This is an example.

```
* dspi_command_data_config_t command;
* DSPI_GetDefaultDataCommandConfig(&command);
*
```

## Parameters

|                |                                                                   |
|----------------|-------------------------------------------------------------------|
| <i>command</i> | Pointer to the <code>dspi_command_data_config_t</code> structure. |
|----------------|-------------------------------------------------------------------|

### 11.2.7.29 void DSPI\_MasterWriteDataBlocking ( SPI\_Type \* *base*,                           *dspi\_command\_data\_config\_t* \* *command*, uint16\_t *data* )

In master mode, the 16-bit data is appended to the 16-bit command info. The command portion provides characteristics of the data, such as the optional continuous chip select operation between transfers, the desired Clock and Transfer Attributes register to use for the associated SPI frame, the desired PCS signal to use for the data transfer, whether the current transfer is the last in the queue, and whether to clear the transfer count (normally needed when sending the first frame of a data packet). This is an example.

```
* dspi_command_config_t commandConfig;
* commandConfig.isPcsContinuous = true;
* commandConfig.whichCtar = kDSPICtar0;
* commandConfig.whichPcs = kDSPIPcs1;
* commandConfig.clearTransferCount = false;
* commandConfig.isEndOfQueue = false;
* DSPI_MasterWriteDataBlocking(base, &commandConfig, dataWord);
*
```

## Note

This function does not return until after the transmit is complete. Also note that the DSPI must be enabled and running to transmit data (MCR[MDIS] & [HALT] = 0). Because the SPI is a synchronous protocol, the received data is available when the transmit completes.

## Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>base</i>    | DSPI peripheral address.          |
| <i>command</i> | Pointer to the command structure. |
| <i>data</i>    | The data word to be sent.         |

### 11.2.7.30 static uint32\_t DSPI\_MasterGetFormattedCommand ( *dspi\_command\_data\_config\_t* \* *command* ) [inline], [static]

This function allows the caller to pass in the data command structure and returns the command word formatted according to the DSPI PUSHR register bit field placement. The user can then "OR" the returned command word with the desired data to send and use the function **DSPI\_HAL\_WriteCommand-DataMastermode** or **DSPI\_HAL\_WriteCommandDataMastermodeBlocking** to write the entire 32-bit command data word to the PUSHR. This helps improve performance in cases where the command structure is constant. For example, the user calls this function before starting a transfer to generate the command word. When they are ready to transmit the data, they OR this formatted command word with

the desired data to transmit. This process increases transmit performance when compared to calling send functions, such as **DSPI\_HAL\_WriteDataMastermode**, which format the command word each time a data word is to be sent.

#### Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>command</i> | Pointer to the command structure. |
|----------------|-----------------------------------|

#### Returns

The command word formatted to the PUSHR data register bit field.

### 11.2.7.31 void DSPI\_MasterWriteCommandDataBlocking ( SPI\_Type \* *base*, uint32\_t *data* )

In this function, the user must append the 16-bit data to the 16-bit command information and then provide the total 32-bit word as the data to send. The command portion provides characteristics of the data, such as the optional continuous chip select operation between transfers, the desired Clock and Transfer Attributes register to use for the associated SPI frame, the desired PCS signal to use for the data transfer, whether the current transfer is the last in the queue, and whether to clear the transfer count (normally needed when sending the first frame of a data packet). The user is responsible for appending this command with the data to send. This is an example:

```
* dataWord = <16-bit command> | <16-bit data>;
* DSPI_MasterWriteCommandDataBlocking(base, dataWord);
*
```

#### Note

This function does not return until after the transmit is complete. Also note that the DSPI must be enabled and running to transmit data (MCR[MDIS] & [HALT] = 0). Because the SPI is a synchronous protocol, the received data is available when the transmit completes.

For a blocking polling transfer, see methods below.

| Option 1                                                             |
|----------------------------------------------------------------------|
| uint32_t command_to_send = DSPI_MasterGetFormattedCommand(&command); |
| uint32_t data0 = command_to_send   data_need_to_send_0;              |
| uint32_t data1 = command_to_send   data_need_to_send_1;              |
| uint32_t data2 = command_to_send   data_need_to_send_2;              |
|                                                                      |
| DSPI_MasterWriteCommandDataBlocking(base,data0);                     |
| DSPI_MasterWriteCommandDataBlocking(base,data1);                     |
| DSPI_MasterWriteCommandDataBlocking(base,data2);                     |

**Option 2**

|                                                                  |
|------------------------------------------------------------------|
| DSPI_MasterWriteDataBlocking(base,&command,data_need_to_send_0); |
| DSPI_MasterWriteDataBlocking(base,&command,data_need_to_send_1); |
| DSPI_MasterWriteDataBlocking(base,&command,data_need_to_send_2); |

Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | DSPI peripheral address.                              |
| <i>data</i> | The data word (command and data combined) to be sent. |

#### **11.2.7.32 static void DSPI\_SlaveWriteData ( SPI\_Type \* *base*, uint32\_t *data* ) [inline], [static]**

In slave mode, up to 16-bit words may be written.

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | DSPI peripheral address. |
| <i>data</i> | The data to send.        |

#### **11.2.7.33 void DSPI\_SlaveWriteDataBlocking ( SPI\_Type \* *base*, uint32\_t *data* )**

In slave mode, up to 16-bit words may be written. The function first clears the transmit complete flag, writes data into data register, and finally waits until the data is transmitted.

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | DSPI peripheral address. |
| <i>data</i> | The data to send.        |

#### **11.2.7.34 static uint32\_t DSPI\_ReadData ( SPI\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The data from the read data buffer.

#### 11.2.7.35 void DSPI\_SetDummyData ( SPI\_Type \* *base*, uint8\_t *dummyData* )

Parameters

|                  |                                                |
|------------------|------------------------------------------------|
| <i>base</i>      | DSPI peripheral address.                       |
| <i>dummyData</i> | Data to be transferred when tx buffer is NULL. |

#### 11.2.7.36 void DSPI\_MasterTransferCreateHandle ( SPI\_Type \* *base*, dspi\_master\_handle\_t \* *handle*, dspi\_master\_transfer\_callback\_t *callback*, void \* *userData* )

This function initializes the DSPI handle, which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, call this API once to get the initialized handle.

Parameters

|                 |                                                              |
|-----------------|--------------------------------------------------------------|
| <i>base</i>     | DSPI peripheral base address.                                |
| <i>handle</i>   | DSPI handle pointer to <a href="#">_dspi_master_handle</a> . |
| <i>callback</i> | DSPI callback.                                               |
| <i>userData</i> | Callback function parameter.                                 |

#### 11.2.7.37 status\_t DSPI\_MasterTransferBlocking ( SPI\_Type \* *base*, dspi\_transfer\_t \* *transfer* )

This function transfers data using polling. This is a blocking function, which does not return until all transfers have been completed.

Parameters

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>base</i>     | DSPI peripheral base address.                             |
| <i>transfer</i> | Pointer to the <a href="#">dspi_transfer_t</a> structure. |

Returns

status of status\_t.

#### 11.2.7.38 status\_t DSPI\_MasterTransferNonBlocking ( SPI\_Type \* *base*,                   dspi\_master\_handle\_t \* *handle*, dspi\_transfer\_t \* *transfer* )

This function transfers data using interrupts. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Parameters

|                 |                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>     | DSPI peripheral base address.                                                                 |
| <i>handle</i>   | Pointer to the <a href="#">_dspi_master_handle</a> structure which stores the transfer state. |
| <i>transfer</i> | Pointer to the <a href="#">dspi_transfer_t</a> structure.                                     |

Returns

status of status\_t.

#### 11.2.7.39 status\_t DSPI\_MasterHalfDuplexTransferBlocking ( SPI\_Type \* *base*,                   dspi\_half\_duplex\_transfer\_t \* *xfer* )

This function will do a half-duplex transfer for DSPI master. This is a blocking function, which does not return until all transfer have been completed. And data transfer will be half-duplex, users can set transmit first or receive first.

Parameters

|             |                                                                  |
|-------------|------------------------------------------------------------------|
| <i>base</i> | DSPI base pointer                                                |
| <i>xfer</i> | pointer to <a href="#">dspi_half_duplex_transfer_t</a> structure |

Returns

status of status\_t.

#### 11.2.7.40 status\_t DSPI\_MasterHalfDuplexTransferNonBlocking ( SPI\_Type \* *base*, dspi\_master\_handle\_t \* *handle*, dspi\_half\_duplex\_transfer\_t \* *xfer* )

This function transfers data using interrupts, the transfer mechanism is half-duplex. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Parameters

|               |                                                                                       |
|---------------|---------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                         |
| <i>handle</i> | pointer to <code>_dspi_master_handle</code> structure which stores the transfer state |
| <i>xfer</i>   | pointer to <code>dspi_half_duplex_transfer_t</code> structure                         |

Returns

status of status\_t.

#### 11.2.7.41 `status_t DSPI_MasterTransferGetCount ( SPI_Type * base, dspi_master_handle_t * handle, size_t * count )`

This function gets the master transfer count.

Parameters

|               |                                                                                            |
|---------------|--------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                              |
| <i>handle</i> | Pointer to the <code>_dspi_master_handle</code> 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.

#### 11.2.7.42 `void DSPI_MasterTransferAbort ( SPI_Type * base, dspi_master_handle_t * handle )`

This function aborts a transfer using an interrupt.

Parameters

|               |                                                                                            |
|---------------|--------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                              |
| <i>handle</i> | Pointer to the <code>_dspi_master_handle</code> structure which stores the transfer state. |

#### 11.2.7.43 `void DSPI_MasterTransferHandleIRQ ( SPI_Type * base, dspi_master_handle_t * handle )`

This function processes the DSPI transmit and receive IRQ.

Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                                 |
| <i>handle</i> | Pointer to the <a href="#">_dspi_master_handle</a> structure which stores the transfer state. |

#### 11.2.7.44 void DSPI\_SlaveTransferCreateHandle ( SPI\_Type \* *base*, dspi\_slave\_handle\_t \* *handle*, dspi\_slave\_transfer\_callback\_t *callback*, void \* *userData* )

This function initializes the DSPI handle, which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, call this API once to get the initialized handle.

Parameters

|                 |                                                                 |
|-----------------|-----------------------------------------------------------------|
| <i>handle</i>   | DSPI handle pointer to the <a href="#">_dspi_slave_handle</a> . |
| <i>base</i>     | DSPI peripheral base address.                                   |
| <i>callback</i> | DSPI callback.                                                  |
| <i>userData</i> | Callback function parameter.                                    |

#### 11.2.7.45 status\_t DSPI\_SlaveTransferNonBlocking ( SPI\_Type \* *base*, dspi\_slave\_handle\_t \* *handle*, dspi\_transfer\_t \* *transfer* )

This function transfers data using an interrupt. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Parameters

|                 |                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------|
| <i>base</i>     | DSPI peripheral base address.                                                                |
| <i>handle</i>   | Pointer to the <a href="#">_dspi_slave_handle</a> structure which stores the transfer state. |
| <i>transfer</i> | Pointer to the <a href="#">dspi_transfer_t</a> structure.                                    |

Returns

status of [status\\_t](#).

#### 11.2.7.46 status\_t DSPI\_SlaveTransferGetCount ( SPI\_Type \* *base*, dspi\_slave\_handle\_t \* *handle*, size\_t \* *count* )

This function gets the slave transfer count.

Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                                 |
| <i>handle</i> | Pointer to the <a href="#">_dspi_master_handle</a> 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.

#### 11.2.7.47 void DSPI\_SlaveTransferAbort ( SPI\_Type \* *base*, dspi\_slave\_handle\_t \* *handle* )

This function aborts a transfer using an interrupt.

Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                                |
| <i>handle</i> | Pointer to the <a href="#">_dspi_slave_handle</a> structure which stores the transfer state. |

#### 11.2.7.48 void DSPI\_SlaveTransferHandleIRQ ( SPI\_Type \* *base*, dspi\_slave\_handle\_t \* *handle* )

This function processes the DSPI transmit and receive IRQ.

Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                                |
| <i>handle</i> | Pointer to the <a href="#">_dspi_slave_handle</a> structure which stores the transfer state. |

#### 11.2.7.49 uint8\_t DSPI\_GetDummyDataInstance ( SPI\_Type \* *base* )

The purpose of this API is to avoid MISRA rule8.5 : Multiple declarations of externally-linked object or function [g\\_dspiDummyData](#).

param base DSPI peripheral base address.

### 11.2.8 Variable Documentation

#### 11.2.8.1 volatile uint8\_t g\_dspiDummyData[]

## 11.3 DSPI eDMA Driver

### 11.3.1 Overview

This section describes the programming interface of the DSPI peripheral driver. The DSPI driver configures DSPI module and provides functional and transactional interfaces to build the DSPI application.

## Data Structures

- struct [dspi\\_master\\_edma\\_handle\\_t](#)  
*DSPI master eDMA transfer handle structure used for the transactional API. [More...](#)*
- struct [dspi\\_slave\\_edma\\_handle\\_t](#)  
*DSPI slave eDMA transfer handle structure used for the transactional API. [More...](#)*

## Macros

- #define [DSPI\\_EDMA\\_MAX\\_TRANSFER\\_SIZE](#)(base, width)  
*DSPI EDMA max transfer data size calculate.*

## Typedefs

- typedef void(\* [dspi\\_master\\_edma\\_transfer\\_callback\\_t](#) )(SPI\_Type \*base, dsPICORE4\_EdmaHandle\_t \*handle, [status\\_t](#) status, void \*userData)  
*Completion callback function pointer type.*
- typedef void(\* [dsPICORE4\\_EdmaHandle\\_t](#) )(SPI\_Type \*base, dsPICORE4\_EdmaHandle\_t \*handle, [status\\_t](#) status, void \*userData)  
*Completion callback function pointer type.*

## Driver version

- #define [FSL\\_DSPI\\_EDMA\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 2, 4))  
*DSPI EDMA driver version 2.2.4.*

## Transactional APIs

- void [DSPI\\_MasterTransferCreateHandleEDMA](#) (SPI\_Type \*base, dsPICORE4\_EdmaHandle\_t \*handle, [dsPICORE4\\_EdmaHandle\\_t](#) \*edmaRxRegToRxDataHandle, [dsPICORE4\\_EdmaHandle\\_t](#) \*edmaTxDataToIntermediaryHandle, [dsPICORE4\\_EdmaHandle\\_t](#) \*edmaIntermediaryToTxRegHandle)  
*Initializes the DSPI master eDMA handle.*
- [status\\_t DSPI\\_MasterTransferEDMA](#) (SPI\_Type \*base, dsPICORE4\_EdmaHandle\_t \*handle, [dsPICORE4\\_EdmaHandle\\_t](#) \*transfer)  
*DSPI master transfer data using eDMA.*

- `status_t DSPI_MasterHalfDuplexTransferEDMA` (`SPI_Type *base, dspi_master_edma_handle_t *handle, dspi_half_duplex_transfer_t *xfer`)
 

*Transfers a block of data using a eDMA method.*
- `void DSPI_MasterTransferAbortEDMA` (`SPI_Type *base, dspi_master_edma_handle_t *handle`)
 

*DSPI master aborts a transfer which is using eDMA.*
- `status_t DSPI_MasterTransferGetCountEDMA` (`SPI_Type *base, dspi_master_edma_handle_t *handle, size_t *count`)
 

*Gets the master eDMA transfer count.*
- `void DSPI_SlaveTransferCreateHandleEDMA` (`SPI_Type *base, dspi_slave_edma_handle_t *handle, dspi_slave_edma_transfer_callback_t callback, void *userData, edma_handle_t *edmaRxRegToRxDataHandle, edma_handle_t *edmaTxDataToTxRegHandle`)
 

*Initializes the DSPI slave eDMA handle.*
- `status_t DSPI_SlaveTransferEDMA` (`SPI_Type *base, dspi_slave_edma_handle_t *handle, dspi_transfer_t *transfer`)
 

*DSPI slave transfer data using eDMA.*
- `void DSPI_SlaveTransferAbortEDMA` (`SPI_Type *base, dspi_slave_edma_handle_t *handle`)
 

*DSPI slave aborts a transfer which is using eDMA.*
- `status_t DSPI_SlaveTransferGetCountEDMA` (`SPI_Type *base, dspi_slave_edma_handle_t *handle, size_t *count`)
 

*Gets the slave eDMA transfer count.*

### 11.3.2 Data Structure Documentation

#### 11.3.2.1 struct \_dspi\_master\_edma\_handle

Forward declaration of the DSPI eDMA master handle typedefs.

#### Data Fields

- `uint32_t bitsPerFrame`

*The desired number of bits per frame.*
- `volatile uint32_t command`

*The desired data command.*
- `volatile uint32_t lastCommand`

*The desired last data command.*
- `uint8_t fifoSize`

*FIFO dataSize.*
- `volatile bool isPcsActiveAfterTransfer`

*Indicates whether the PCS signal keeps active after the last frame transfer.*
- `uint8_t nbytes`

*eDMA minor byte transfer count initially configured.*
- `volatile uint8_t state`

*DSPI transfer state, see `_dspi_transfer_state`.*
- `uint8_t *volatile txData`

*Send buffer.*
- `uint8_t *volatile rxData`

*Receive buffer.*

- volatile size\_t **remainingSendByteCount**  
*A number of bytes remaining to send.*
- volatile size\_t **remainingReceiveByteCount**  
*A number of bytes remaining to receive.*
- size\_t **totalByteCount**  
*A number of transfer bytes.*
- uint32\_t **rxBuffIfNull**  
*Used if there is not rxData for DMA purpose.*
- uint32\_t **txBuffIfNull**  
*Used if there is not txData for DMA purpose.*
- dspi\_master\_edma\_transfer\_callback\_t **callback**  
*Completion callback.*
- void \* **userData**  
*Callback user data.*
- edma\_handle\_t \* **edmaRxRegToRxDataHandle**  
*edma\_handle\_t handle point used for RxReg to RxData buff*
- edma\_handle\_t \* **edmaTxDataToIntermediaryHandle**  
*edma\_handle\_t handle point used for TxData to Intermediary*
- edma\_handle\_t \* **edmaIntermediaryToTxRegHandle**  
*edma\_handle\_t handle point used for Intermediary to TxReg*
- edma\_tcd\_t **dspiSoftwareTCD** [2]  
*SoftwareTCD , internal used.*

## Field Documentation

- (1) uint32\_t **dspi\_master\_edma\_handle\_t::bitsPerFrame**
- (2) volatile uint32\_t **dspi\_master\_edma\_handle\_t::command**
- (3) volatile uint32\_t **dspi\_master\_edma\_handle\_t::lastCommand**
- (4) uint8\_t **dspi\_master\_edma\_handle\_t::fifoSize**
- (5) volatile bool **dspi\_master\_edma\_handle\_t::isPcsActiveAfterTransfer**
- (6) uint8\_t **dspi\_master\_edma\_handle\_t::nbytes**
- (7) volatile uint8\_t **dspi\_master\_edma\_handle\_t::state**
- (8) uint8\_t\* volatile **dspi\_master\_edma\_handle\_t::txData**
- (9) uint8\_t\* volatile **dspi\_master\_edma\_handle\_t::rxData**
- (10) volatile size\_t **dspi\_master\_edma\_handle\_t::remainingSendByteCount**
- (11) volatile size\_t **dspi\_master\_edma\_handle\_t::remainingReceiveByteCount**
- (12) uint32\_t **dspi\_master\_edma\_handle\_t::rxBuffIfNull**
- (13) uint32\_t **dspi\_master\_edma\_handle\_t::txBuffIfNull**

- (14) `dspi_master_edma_transfer_callback_t dspi_master_edma_handle_t::callback`
- (15) `void* dspi_master_edma_handle_t::userData`

### 11.3.2.2 struct \_dspi\_slave\_edma\_handle

Forward declaration of the DSPI eDMA slave handle typedefs.

#### Data Fields

- `uint32_t bitsPerFrame`  
*The desired number of bits per frame.*
- `uint8_t *volatile txData`  
*Send buffer.*
- `uint8_t *volatile rxData`  
*Receive buffer.*
- `volatile size_t remainingSendByteCount`  
*A number of bytes remaining to send.*
- `volatile size_t remainingReceiveByteCount`  
*A number of bytes remaining to receive.*
- `size_t totalByteCount`  
*A number of transfer bytes.*
- `uint32_t rxBuffIfNull`  
*Used if there is not rxData for DMA purpose.*
- `uint32_t txBuffIfNull`  
*Used if there is not txData for DMA purpose.*
- `uint32_t txLastData`  
*Used if there is an extra byte when 16bits per frame for DMA purpose.*
- `uint8_t nbytes`  
*eDMA minor byte transfer count initially configured.*
- `volatile uint8_t state`  
*DSPI transfer state.*
- `dspi_slave_edma_transfer_callback_t callback`  
*Completion callback.*
- `void *userData`  
*Callback user data.*
- `edma_handle_t * edmaRxRegToRxDataHandle`  
*edma\_handle\_t handle point used for RxReg to RxData buff*
- `edma_handle_t * edmaTxDataToTxRegHandle`  
*edma\_handle\_t handle point used for TxData to TxReg*

#### Field Documentation

- (1) `uint32_t dspi_slave_edma_handle_t::bitsPerFrame`
- (2) `uint8_t* volatile dspi_slave_edma_handle_t::txData`
- (3) `uint8_t* volatile dspi_slave_edma_handle_t::rxData`
- (4) `volatile size_t dspi_slave_edma_handle_t::remainingSendByteCount`

- (5) volatile size\_t dspi\_slave\_edma\_handle\_t::remainingReceiveByteCount
- (6) uint32\_t dspi\_slave\_edma\_handle\_t::rxBuffIfNull
- (7) uint32\_t dspi\_slave\_edma\_handle\_t::txBuffIfNull
- (8) uint32\_t dspi\_slave\_edma\_handle\_t::txLastData
- (9) uint8\_t dspi\_slave\_edma\_handle\_t::nbytes
- (10) volatile uint8\_t dspi\_slave\_edma\_handle\_t::state
- (11) dspi\_slave\_edma\_transfer\_callback\_t dspi\_slave\_edma\_handle\_t::callback
- (12) void\* dspi\_slave\_edma\_handle\_t::userData

### 11.3.3 Macro Definition Documentation

#### 11.3.3.1 #define DSPI\_EDMA\_MAX\_TRANSFER\_SIZE( *base*, *width* )

**Value:**

```
((1 == FSL_FEATURE_DSPI_HAS_SEPARATE_DMA_RX_TX_REQn(base)) ? ((width > 8U) ? 65534U : 32767U) : \
((width > 8U) ? 1022U : 511U))
```

Parameters

|              |                               |
|--------------|-------------------------------|
| <i>base</i>  | DSPI peripheral base address. |
| <i>width</i> | Transfer width                |

### 11.3.4 Typedef Documentation

#### 11.3.4.1 typedef void(\* dspi\_master\_edma\_transfer\_callback\_t)(SPI\_Type \*base, dspi\_master\_edma\_handle\_t \*handle, status\_t status, void \*userData)

Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                |
| <i>handle</i> | A pointer to the handle for the DSPI master. |

|                 |                                                                   |
|-----------------|-------------------------------------------------------------------|
| <i>status</i>   | Success or error code describing whether the transfer completed.  |
| <i>userData</i> | An arbitrary pointer-dataSized value passed from the application. |

### 11.3.4.2 **typedef void(\* dspi\_slave\_edma\_transfer\_callback\_t)(SPI\_Type \*base, dspi\_slave\_edma\_handle\_t \*handle, status\_t status, void \*userData)**

Parameters

|                 |                                                                   |
|-----------------|-------------------------------------------------------------------|
| <i>base</i>     | DSPI peripheral base address.                                     |
| <i>handle</i>   | A pointer to the handle for the DSPI slave.                       |
| <i>status</i>   | Success or error code describing whether the transfer completed.  |
| <i>userData</i> | An arbitrary pointer-dataSized value passed from the application. |

## 11.3.5 Function Documentation

### 11.3.5.1 **void DSPI\_MasterTransferCreateHandleEDMA ( SPI\_Type \* *base*, dspi\_master\_edma\_handle\_t \* *handle*, dspi\_master\_edma\_transfer\_callback\_t *callback*, void \* *userData*, edma\_handle\_t \* *edmaRxRegToRxDataHandle*, edma\_handle\_t \* *edmaTxDataToIntermediaryHandle*, edma\_handle\_t \* *edmaIntermediaryToTxRegHandle* )**

This function initializes the DSPI eDMA handle which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, call this API once to get the initialized handle.

Note

DSPI eDMA has separated (RX and TX as two sources) or shared (RX and TX are the same source) DMA request source.

- For the separated DMA request source, enable and set the RX DMAMUX source for edmaRxRegToRxDataHandle and TX DMAMUX source for edmaIntermediaryToTxRegHandle.
- For the shared DMA request source, enable and set the RX/RX DMAMUX source for the edmaRxRegToRxDataHandle.

Parameters

---

|                                          |                                                                           |
|------------------------------------------|---------------------------------------------------------------------------|
| <i>base</i>                              | DSPI peripheral base address.                                             |
| <i>handle</i>                            | DSPI handle pointer to <a href="#">_dspl_master_edma_handle</a> .         |
| <i>callback</i>                          | DSPI callback.                                                            |
| <i>userData</i>                          | A callback function parameter.                                            |
| <i>edmaRxRegTo-RxDataHandle</i>          | edmaRxRegToRxDataHandle pointer to <a href="#">edma_handle_t</a> .        |
| <i>edmaTxData-To-Intermediary-Handle</i> | edmaTxDataToIntermediaryHandle pointer to <a href="#">edma_handle_t</a> . |
| <i>edma-Intermediary-ToTxReg-Handle</i>  | edmaIntermediaryToTxRegHandle pointer to <a href="#">edma_handle_t</a> .  |

### 11.3.5.2 status\_t DSPI\_MasterTransferEDMA ( SPI\_Type \* *base*, dspi-master\_edma\_handle\_t \* *handle*, dspi\_transfer\_t \* *transfer* )

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

#### Note

The max transfer size of each transfer depends on whether the instance's Tx/Rx shares the same DMA request. If **FSL\_FEATURE\_DSPI\_HAS\_SEPARATE\_DMA\_RX\_TX\_REQn(x)** is true, then the max transfer size is 32767 datawidth of data, otherwise is 511.

#### Parameters

|                 |                                                                                                      |
|-----------------|------------------------------------------------------------------------------------------------------|
| <i>base</i>     | DSPI peripheral base address.                                                                        |
| <i>handle</i>   | A pointer to the <a href="#">_dspl_master_edma_handle</a> structure which stores the transfer state. |
| <i>transfer</i> | A pointer to the <a href="#">dspi_transfer_t</a> structure.                                          |

#### Returns

status of status\_t.

### 11.3.5.3 status\_t DSPI\_MasterHalfDuplexTransferEDMA ( SPI\_Type \* *base*, dspi\_master\_edma\_handle\_t \* *handle*, dspi\_half\_duplex\_transfer\_t \* *xfer* )

This function transfers data using eDNA, the transfer mechanism is half-duplex. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Parameters

|               |                                                                                                             |
|---------------|-------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI base pointer                                                                                           |
| <i>handle</i> | A pointer to the <a href="#"><u>_dspi_master_edma_handle</u></a> structure which stores the transfer state. |
| <i>xfer</i>   | A pointer to the <a href="#"><u>dspi_half_duplex_transfer_t</u></a> structure.                              |

Returns

status of status\_t.

#### **11.3.5.4 void DSPI\_MasterTransferAbortEDMA ( SPI\_Type \* *base*,                           dspi\_master\_edma\_handle\_t \* *handle* )**

This function aborts a transfer which is using eDMA.

Parameters

|               |                                                                                                             |
|---------------|-------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                                               |
| <i>handle</i> | A pointer to the <a href="#"><u>_dspi_master_edma_handle</u></a> structure which stores the transfer state. |

#### **11.3.5.5 status\_t DSPI\_MasterTransferGetCountEDMA ( SPI\_Type \* *base*,                           dspi\_master\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

This function gets the master eDMA transfer count.

Parameters

|               |                                                                                                             |
|---------------|-------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                                               |
| <i>handle</i> | A pointer to the <a href="#"><u>_dspi_master_edma_handle</u></a> structure which stores the transfer state. |
| <i>count</i>  | A number of bytes transferred by the non-blocking transaction.                                              |

Returns

status of status\_t.

#### **11.3.5.6 void DSPI\_SlaveTransferCreateHandleEDMA ( SPI\_Type \* *base*,                           dspi\_slave\_edma\_handle\_t \* *handle*, dspi\_slave\_edma\_transfer\_callback\_t                           *callback*, void \* *userData*, edma\_handle\_t \* *edmaRxRegToRxDataHandle*,                           edma\_handle\_t \* *edmaTxDataToTxRegHandle* )**

This function initializes the DSPI eDMA handle which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, call this API once to get the initialized handle.

## Note

DSPI eDMA has separated (RN and TX in 2 sources) or shared (RX and TX are the same source) DMA request source.

- For the separated DMA request source, enable and set the RX DMAMUX source for edmaRxRegToRxDataHandle and TX DMAMUX source for edmaTxDataToTxRegHandle.
- For the shared DMA request source, enable and set the RX/RX DMAMUX source for the edmaRxRegToRxDataHandle.

## Parameters

|                                |                                                                    |
|--------------------------------|--------------------------------------------------------------------|
| <i>base</i>                    | DSPI peripheral base address.                                      |
| <i>handle</i>                  | DSPI handle pointer to <a href="#">_dspl_slave_edma_handle</a> .   |
| <i>callback</i>                | DSPI callback.                                                     |
| <i>userData</i>                | A callback function parameter.                                     |
| <i>edmaRxRegToRxDataHandle</i> | edmaRxRegToRxDataHandle pointer to <a href="#">edma_handle_t</a> . |
| <i>edmaTxDataToTxRegHandle</i> | edmaTxDataToTxRegHandle pointer to <a href="#">edma_handle_t</a> . |

#### 11.3.5.7 **status\_t DSPI\_SlaveTransferEDMA ( SPI\_Type \* *base*, dspi\_slave\_edma\_handle\_t \* *handle*, dspi\_transfer\_t \* *transfer* )**

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called. Note that the slave eDMA transfer doesn't support transfer\_size is 1 when the bitsPerFrame is greater than eight.

## Note

The max transfer size of each transfer depends on whether the instance's Tx/Rx shares the same DMA request. If **FSL\_FEATURE\_DSPI\_HAS\_SEPARATE\_DMA\_RX\_TX\_REQn(x)** is true, then the max transfer size is 32767 datawidth of data, otherwise is 511.

## Parameters

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

|                 |                                                                                                            |
|-----------------|------------------------------------------------------------------------------------------------------------|
| <i>handle</i>   | A pointer to the <a href="#"><u>_dspl_slave_edma_handle</u></a> structure which stores the transfer state. |
| <i>transfer</i> | A pointer to the <a href="#"><u>dspl_transfer_t</u></a> structure.                                         |

Returns

status of status\_t.

#### **11.3.5.8 void DSPI\_SlaveTransferAbortEDMA ( SPI\_Type \* *base*, dspl\_slave\_edma\_handle\_t \* *handle* )**

This function aborts a transfer which is using eDMA.

Parameters

|               |                                                                                                            |
|---------------|------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                                              |
| <i>handle</i> | A pointer to the <a href="#"><u>_dspl_slave_edma_handle</u></a> structure which stores the transfer state. |

#### **11.3.5.9 status\_t DSPI\_SlaveTransferGetCountEDMA ( SPI\_Type \* *base*, dspl\_slave\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

This function gets the slave eDMA transfer count.

Parameters

|               |                                                                                                            |
|---------------|------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                                              |
| <i>handle</i> | A pointer to the <a href="#"><u>_dspl_slave_edma_handle</u></a> structure which stores the transfer state. |
| <i>count</i>  | A number of bytes transferred so far by the non-blocking transaction.                                      |

Returns

status of status\_t.

## 11.4 DSPI FreeRTOS Driver

### 11.4.1 Overview

#### Driver version

- #define `FSL_DSPI_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 2, 4)`)  
*DSPI FreeRTOS driver version 2.2.4.*

#### DSPI RTOS Operation

- `status_t DSPI_RTOS_Init` (`dspi_rtos_handle_t *handle, SPI_Type *base, const dspi_master_config_t *masterConfig, uint32_t srcClock_Hz`)  
*Initializes the DSPI.*
- `status_t DSPI_RTOS_Deinit` (`dspi_rtos_handle_t *handle`)  
*Deinitializes the DSPI.*
- `status_t DSPI_RTOS_Transfer` (`dspi_rtos_handle_t *handle, dspi_transfer_t *transfer`)  
*Performs the SPI transfer.*

### 11.4.2 Macro Definition Documentation

#### 11.4.2.1 #define `FSL_DSPI_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 2, 4)`)

### 11.4.3 Function Documentation

#### 11.4.3.1 `status_t DSPI_RTOS_Init ( dspi_rtos_handle_t * handle, SPI_Type * base, const dspi_master_config_t * masterConfig, uint32_t srcClock_Hz )`

This function initializes the DSPI module and the related RTOS context.

Parameters

|                           |                                                                           |
|---------------------------|---------------------------------------------------------------------------|
| <code>handle</code>       | The RTOS DSPI handle, the pointer to an allocated space for RTOS context. |
| <code>base</code>         | The pointer base address of the DSPI instance to initialize.              |
| <code>masterConfig</code> | A configuration structure to set-up the DSPI in master mode.              |
| <code>srcClock_Hz</code>  | A frequency of the input clock of the DSPI module.                        |

Returns

status of the operation.

#### 11.4.3.2 status\_t DSPI\_RTOS\_Deinit ( *dspi\_rtos\_handle\_t \* handle* )

This function deinitializes the DSPI module and the related RTOS context.

Parameters

|               |                       |
|---------------|-----------------------|
| <i>handle</i> | The RTOS DSPI handle. |
|---------------|-----------------------|

#### **11.4.3.3 status\_t DSPI\_RTOS\_Transfer ( *dspi\_rtos\_handle\_t* \* *handle*, *dspi\_transfer\_t* \* *transfer* )**

This function performs the SPI transfer according to the data given in the transfer structure.

Parameters

|                 |                                                 |
|-----------------|-------------------------------------------------|
| <i>handle</i>   | The RTOS DSPI handle.                           |
| <i>transfer</i> | A structure specifying the transfer parameters. |

Returns

status of the operation.

## 11.5 DSPI CMSIS Driver

This section describes the programming interface of the DSPI Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method see <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

### 11.5.1 Function groups

#### 11.5.1.1 DSPI CMSIS GetVersion Operation

This function group will return the DSPI CMSIS Driver version to user.

#### 11.5.1.2 DSPI CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

#### 11.5.1.3 DSPI CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialized the instance in master mode or slave mode. And this API must be called before you configure an instance or after you Deinit an instance. The right steps to start an instance is that you must initialize the instance which been selected firstly, then you can power on the instance. After these all have been done, you can configure the instance by using control operation. If you want to Uninitialize the instance, you must power off the instance first.

#### 11.5.1.4 DSPI CMSIS Transfer Operation

This function group controls the transfer, master send/receive data, and slave send/receive data.

#### 11.5.1.5 DSPI CMSIS Status Operation

This function group gets the DSPI transfer status.

#### 11.5.1.6 DSPI CMSIS Control Operation

This function can configure instance as master mode or slave mode, set baudrate for master mode transfer, get current baudrate of master mode transfer, set transfer data bits and other control command.

## 11.5.2 Typical use case

### 11.5.2.1 Master Operation

```
/* Variables */
uint8_t masterRxData[TRANSFER_SIZE] = {0U};
uint8_t masterTxData[TRANSFER_SIZE] = {0U};

/*DSPI master init*/
Driver_SPI0.Initialize(DSPI_MasterSignalEvent_t);
Driver_SPI0.PowerControl(ARM_POWER_FULL);
Driver_SPI0.Control(ARM_SPI_MODE_MASTER, TRANSFER_BAUDRATE);

/* Start master transfer */
Driver_SPI0.Transfer(masterTxData, masterRxData, TRANSFER_SIZE);

/* Master power off */
Driver_SPI0.PowerControl(ARM_POWER_OFF);

/* Master uninitialize */
Driver_SPI0.Uninitialize();
```

### 11.5.2.2 Slave Operation

```
/* Variables */
uint8_t slaveRxData[TRANSFER_SIZE] = {0U};
uint8_t slaveTxData[TRANSFER_SIZE] = {0U};

/*DSPI slave init*/
Driver_SPI1.Initialize(DSPI_SlaveSignalEvent_t);
Driver_SPI1.PowerControl(ARM_POWER_FULL);
Driver_SPI1.Control(ARM_SPI_MODE_SLAVE, false);

/* Start slave transfer */
Driver_SPI1.Transfer(slaveTxData, slaveRxData, TRANSFER_SIZE);

/* slave power off */
Driver_SPI1.PowerControl(ARM_POWER_OFF);

/* slave uninitialize */
Driver_SPI1.Uninitialize();
```

# Chapter 12

## eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

### 12.1 Overview

The MCUXpresso SDK provides a peripheral driver for the enhanced Direct Memory Access (eDMA) of MCUXpresso SDK devices.

### 12.2 Typical use case

#### 12.2.1 eDMA Operation

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/edma

## Data Structures

- struct `edma_config_t`  
*eDMA global configuration structure. [More...](#)*
- struct `edma_transfer_config_t`  
*eDMA transfer configuration [More...](#)*
- struct `edma_channel_Preemption_config_t`  
*eDMA channel priority configuration [More...](#)*
- struct `edma_minor_offset_config_t`  
*eDMA minor offset configuration [More...](#)*
- struct `edma_tcd_t`  
*eDMA TCD. [More...](#)*
- struct `edma_handle_t`  
*eDMA transfer handle structure [More...](#)*

## Macros

- #define `DMA_DCHPRI_INDEX`(channel) (((channel) & ~0x03U) | (3U - ((channel)&0x03U)))  
*Compute the offset unit from DCHPRI3.*

## Typedefs

- typedef void(\* `edma_callback` )(struct \_edma\_handle \*handle, void \*userData, bool transferDone, uint32\_t tcds)  
*Define callback function for eDMA.*

## Enumerations

- enum `edma_transfer_size_t` {
   
    `kEDMA_TransferSize1Bytes` = 0x0U,
   
    `kEDMA_TransferSize2Bytes` = 0x1U,
   
    `kEDMA_TransferSize4Bytes` = 0x2U,
   
    `kEDMA_TransferSize8Bytes` = 0x3U,
   
    `kEDMA_TransferSize16Bytes` = 0x4U,
   
    `kEDMA_TransferSize32Bytes` = 0x5U }
   
    *eDMA transfer configuration*
- enum `edma_modulo_t` {
   
    `kEDMA_ModuloDisable` = 0x0U,
   
    `kEDMA_Modulo2bytes`,
   
    `kEDMA_Modulo4bytes`,
   
    `kEDMA_Modulo8bytes`,
   
    `kEDMA_Modulo16bytes`,
   
    `kEDMA_Modulo32bytes`,
   
    `kEDMA_Modulo64bytes`,
   
    `kEDMA_Modulo128bytes`,
   
    `kEDMA_Modulo256bytes`,
   
    `kEDMA_Modulo512bytes`,
   
    `kEDMA_Modulo1Kbytes`,
   
    `kEDMA_Modulo2Kbytes`,
   
    `kEDMA_Modulo4Kbytes`,
   
    `kEDMA_Modulo8Kbytes`,
   
    `kEDMA_Modulo16Kbytes`,
   
    `kEDMA_Modulo32Kbytes`,
   
    `kEDMA_Modulo64Kbytes`,
   
    `kEDMA_Modulo128Kbytes`,
   
    `kEDMA_Modulo256Kbytes`,
   
    `kEDMA_Modulo512Kbytes`,
   
    `kEDMA_Modulo1Mbytes`,
   
    `kEDMA_Modulo2Mbytes`,
   
    `kEDMA_Modulo4Mbytes`,
   
    `kEDMA_Modulo8Mbytes`,
   
    `kEDMA_Modulo16Mbytes`,
   
    `kEDMA_Modulo32Mbytes`,
   
    `kEDMA_Modulo64Mbytes`,
   
    `kEDMA_Modulo128Mbytes`,
   
    `kEDMA_Modulo256Mbytes`,
   
    `kEDMA_Modulo512Mbytes`,
   
    `kEDMA_Modulo1Gbytes`,
   
    `kEDMA_Modulo2Gbytes` }
   
    *eDMA modulo configuration*
- enum `edma_bandwidth_t` {

- ```

kEDMA_BandwidthStallNone = 0x0U,
kEDMA_BandwidthStall4Cycle = 0x2U,
kEDMA_BandwidthStall8Cycle = 0x3U }

Bandwidth control.
• enum edma_channel_link_type_t {
    kEDMA_LinkNone = 0x0U,
    kEDMA_MinorLink,
    kEDMA_MajorLink }

Channel link type.
• enum {
    kEDMA_DoneFlag = 0x1U,
    kEDMA_ErrorFlag = 0x2U,
    kEDMA_InterruptFlag = 0x4U }

_edma_channel_status_flags eDMA channel status flags.
• enum {
    kEDMA_DestinationBusErrorFlag = DMA_ES_DBE_MASK,
    kEDMA_SourceBusErrorFlag = DMA_ES_SBE_MASK,
    kEDMA_ScatterGatherErrorFlag = DMA_ES_SGE_MASK,
    kEDMA_NbytesErrorFlag = DMA_ES_NCE_MASK,
    kEDMA_DestinationOffsetErrorFlag = DMA_ES_DOE_MASK,
    kEDMA_DestinationAddressErrorFlag = DMA_ES_DAE_MASK,
    kEDMA_SourceOffsetErrorFlag = DMA_ES_SOE_MASK,
    kEDMA_SourceAddressErrorFlag = DMA_ES_SAE_MASK,
    kEDMA_ErrorChannelFlag = DMA_ES_ERRCHN_MASK,
    kEDMA_ChannelPriorityErrorFlag = DMA_ES_CPE_MASK,
    kEDMA_TransferCanceledFlag = DMA_ES_ECX_MASK,
    kEDMA_ValidFlag = (int)DMA_ES_VLD_MASK }

_edma_error_status_flags eDMA channel error status flags.
• enum edma_interrupt_enable_t {
    kEDMA_ErrorInterruptEnable = 0x1U,
    kEDMA_MajorInterruptEnable = DMA_CSR_INTMAJOR_MASK,
    kEDMA_HalfInterruptEnable = DMA_CSR_INTHALF_MASK }

eDMA interrupt source
• enum edma_transfer_type_t {
    kEDMA_MemoryToMemory = 0x0U,
    kEDMA_PeripheralToMemory,
    kEDMA_MemoryToPeripheral,
    kEDMA_PeripheralToPeripheral }

eDMA transfer type
• enum {
    kStatus_EDMA_QueueFull = MAKE_STATUS(kStatusGroup_EDMA, 0),
    kStatus_EDMA_Busy = MAKE_STATUS(kStatusGroup_EDMA, 1) }

_edma_transfer_status eDMA transfer status

```

Driver version

- #define `FSL_EDMA_DRIVER_VERSION` (`MAKE_VERSION(2, 4, 3)`)

eDMA driver version

eDMA initialization and de-initialization

- void **EDMA_Init** (DMA_Type *base, const **edma_config_t** *config)
Initializes the eDMA peripheral.
- void **EDMA_Deinit** (DMA_Type *base)
Deinitializes the eDMA peripheral.
- void **EDMA_InstallTCD** (DMA_Type *base, uint32_t channel, **edma_tcd_t** *tcd)
Push content of TCD structure into hardware TCD register.
- void **EDMA_GetDefaultConfig** (**edma_config_t** *config)
Gets the eDMA default configuration structure.
- static void **EDMA_EnableContinuousChannelLinkMode** (DMA_Type *base, bool enable)
Enable/Disable continuous channel link mode.
- static void **EDMA_EnableMinorLoopMapping** (DMA_Type *base, bool enable)
Enable/Disable minor loop mapping.

eDMA Channel Operation

- void **EDMA_ResetChannel** (DMA_Type *base, uint32_t channel)
Sets all TCD registers to default values.
- void **EDMA_SetTransferConfig** (DMA_Type *base, uint32_t channel, const **edma_transfer_config_t** *config, **edma_tcd_t** *nextTcd)
Configures the eDMA transfer attribute.
- void **EDMA_SetMinorOffsetConfig** (DMA_Type *base, uint32_t channel, const **edma_minor_offset_config_t** *config)
Configures the eDMA minor offset feature.
- void **EDMA_SetChannelPreemptionConfig** (DMA_Type *base, uint32_t channel, const **edma_channel_Preemption_config_t** *config)
Configures the eDMA channel preemption feature.
- void **EDMA_SetChannelLink** (DMA_Type *base, uint32_t channel, **edma_channel_link_type_t** linkType, uint32_t linkedChannel)
Sets the channel link for the eDMA transfer.
- void **EDMA_SetBandWidth** (DMA_Type *base, uint32_t channel, **edma_bandwidth_t** bandWidth)
Sets the bandwidth for the eDMA transfer.
- void **EDMA_SetModulo** (DMA_Type *base, uint32_t channel, **edma_modulo_t** srcModulo, **edma_modulo_t** destModulo)
Sets the source modulo and the destination modulo for the eDMA transfer.
- static void **EDMA_EnableAsyncRequest** (DMA_Type *base, uint32_t channel, bool enable)
Enables an async request for the eDMA transfer.
- static void **EDMA_EnableAutoStopRequest** (DMA_Type *base, uint32_t channel, bool enable)
Enables an auto stop request for the eDMA transfer.
- void **EDMA_EnableChannelInterrupts** (DMA_Type *base, uint32_t channel, uint32_t mask)
Enables the interrupt source for the eDMA transfer.
- void **EDMA_DisableChannelInterrupts** (DMA_Type *base, uint32_t channel, uint32_t mask)
Disables the interrupt source for the eDMA transfer.
- void **EDMA_SetMajorOffsetConfig** (DMA_Type *base, uint32_t channel, int32_t sourceOffset, int32_t destOffset)
Configures the eDMA channel TCD major offset feature.

eDMA TCD Operation

- void [EDMA_TcdReset](#) (edma_tcd_t *tcd)
Sets all fields to default values for the TCD structure.
- void [EDMA_TcdSetTransferConfig](#) (edma_tcd_t *tcd, const edma_transfer_config_t *config, edma_tcd_t *nextTcd)
Configures the eDMA TCD transfer attribute.
- void [EDMA_TcdSetMinorOffsetConfig](#) (edma_tcd_t *tcd, const edma_minor_offset_config_t *config)
Configures the eDMA TCD minor offset feature.
- void [EDMA_TcdSetChannelLink](#) (edma_tcd_t *tcd, edma_channel_link_type_t linkType, uint32_t linkedChannel)
Sets the channel link for the eDMA TCD.
- static void [EDMA_TcdSetBandWidth](#) (edma_tcd_t *tcd, edma_bandwidth_t bandWidth)
Sets the bandwidth for the eDMA TCD.
- void [EDMA_TcdSetModulo](#) (edma_tcd_t *tcd, edma_modulo_t srcModulo, edma_modulo_t destModulo)
Sets the source modulo and the destination modulo for the eDMA TCD.
- static void [EDMA_TcdEnableAutoStopRequest](#) (edma_tcd_t *tcd, bool enable)
Sets the auto stop request for the eDMA TCD.
- void [EDMA_TcdEnableInterrupts](#) (edma_tcd_t *tcd, uint32_t mask)
Enables the interrupt source for the eDMA TCD.
- void [EDMA_TcdDisableInterrupts](#) (edma_tcd_t *tcd, uint32_t mask)
Disables the interrupt source for the eDMA TCD.
- void [EDMA_TcdSetMajorOffsetConfig](#) (edma_tcd_t *tcd, int32_t sourceOffset, int32_t destOffset)
Configures the eDMA TCD major offset feature.

eDMA Channel Transfer Operation

- static void [EDMA_EnableChannelRequest](#) (DMA_Type *base, uint32_t channel)
Enables the eDMA hardware channel request.
- static void [EDMA_DisableChannelRequest](#) (DMA_Type *base, uint32_t channel)
Disables the eDMA hardware channel request.
- static void [EDMA_TriggerChannelStart](#) (DMA_Type *base, uint32_t channel)
Starts the eDMA transfer by using the software trigger.

eDMA Channel Status Operation

- uint32_t [EDMA_GetRemainingMajorLoopCount](#) (DMA_Type *base, uint32_t channel)
Gets the remaining major loop count from the eDMA current channel TCD.
- static uint32_t [EDMA_GetErrorStatusFlags](#) (DMA_Type *base)
Gets the eDMA channel error status flags.
- uint32_t [EDMA_GetChannelStatusFlags](#) (DMA_Type *base, uint32_t channel)
Gets the eDMA channel status flags.
- void [EDMA_ClearChannelStatusFlags](#) (DMA_Type *base, uint32_t channel, uint32_t mask)
Clears the eDMA channel status flags.

eDMA Transactional Operation

- void [EDMA_CreateHandle](#) (edma_handle_t *handle, DMA_Type *base, uint32_t channel)
Creates the eDMA handle.

- void **EDMA_InstallTCDMemory** (**edma_handle_t** *handle, **edma_tcd_t** *tcdPool, **uint32_t** tcdSize)

Installs the TCDs memory pool into the eDMA handle.
- void **EDMA_SetCallback** (**edma_handle_t** *handle, **edma_callback** callback, void *userData)

Installs a callback function for the eDMA transfer.
- void **EDMA_PrepTransferConfig** (**edma_transfer_config_t** *config, void *srcAddr, **uint32_t** srcWidth, **int16_t** srcOffset, void *destAddr, **uint32_t** destWidth, **int16_t** destOffset, **uint32_t** bytesEachRequest, **uint32_t** transferBytes)

Prepares the eDMA transfer structure configurations.
- void **EDMA_PrepTransfer** (**edma_transfer_config_t** *config, void *srcAddr, **uint32_t** srcWidth, void *destAddr, **uint32_t** destWidth, **uint32_t** bytesEachRequest, **uint32_t** transferBytes, **edma_transfer_type_t** transferType)

Prepares the eDMA transfer structure.
- **status_t EDMA_SubmitTransfer** (**edma_handle_t** *handle, const **edma_transfer_config_t** *config)

Submits the eDMA transfer request.
- void **EDMA_StartTransfer** (**edma_handle_t** *handle)

eDMA starts transfer.
- void **EDMA_StopTransfer** (**edma_handle_t** *handle)

eDMA stops transfer.
- void **EDMA_AbortTransfer** (**edma_handle_t** *handle)

eDMA aborts transfer.
- static **uint32_t EDMA_GetUnusedTCDNumber** (**edma_handle_t** *handle)

Get unused TCD slot number.
- static **uint32_t EDMA_GetNextTCDAddress** (**edma_handle_t** *handle)

Get the next tcd address.
- void **EDMA_HandleIRQ** (**edma_handle_t** *handle)

eDMA IRQ handler for the current major loop transfer completion.

12.3 Data Structure Documentation

12.3.1 struct edma_config_t

Data Fields

- bool **enableContinuousLinkMode**

Enable (true) continuous link mode.
- bool **enableHaltOnError**

Enable (true) transfer halt on error.
- bool **enableRoundRobinArbitration**

Enable (true) round robin channel arbitration method or fixed priority arbitration is used for channel selection.
- bool **enableDebugMode**

Enable(true) eDMA debug mode.

Field Documentation

(1) bool **edma_config_t::enableContinuousLinkMode**

Upon minor loop completion, the channel activates again if that channel has a minor loop channel link enabled and the link channel is itself.

(2) bool edma_config_t::enableHaltOnError

Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

(3) bool edma_config_t::enableDebugMode

When in debug mode, the eDMA stalls the start of a new channel. Executing channels are allowed to complete.

12.3.2 struct edma_transfer_config_t

This structure configures the source/destination transfer attribute.

Data Fields

- **uint32_t srcAddr**
Source data address.
- **uint32_t destAddr**
Destination data address.
- **edma_transfer_size_t srcTransferSize**
Source data transfer size.
- **edma_transfer_size_t destTransferSize**
Destination data transfer size.
- **int16_t srcOffset**
Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.
- **int16_t destOffset**
Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.
- **uint32_t minorLoopBytes**
Bytes to transfer in a minor loop.
- **uint32_t majorLoopCounts**
Major loop iteration count.

Field Documentation**(1) uint32_t edma_transfer_config_t::srcAddr****(2) uint32_t edma_transfer_config_t::destAddr****(3) edma_transfer_size_t edma_transfer_config_t::srcTransferSize****(4) edma_transfer_size_t edma_transfer_config_t::destTransferSize****(5) int16_t edma_transfer_config_t::srcOffset**

- (6) int16_t edma_transfer_config_t::destOffset
- (7) uint32_t edma_transfer_config_t::majorLoopCounts

12.3.3 struct edma_channel_Preemption_config_t

Data Fields

- bool enableChannelPreemption
If true: a channel can be suspended by other channel with higher priority.
- bool enablePreemptAbility
If true: a channel can suspend other channel with low priority.
- uint8_t channelPriority
Channel priority.

12.3.4 struct edma_minor_offset_config_t

Data Fields

- bool enableSrcMinorOffset
Enable(true) or Disable(false) source minor loop offset.
- bool enableDestMinorOffset
Enable(true) or Disable(false) destination minor loop offset.
- uint32_t minorOffset
Offset for a minor loop mapping.

Field Documentation

- (1) bool edma_minor_offset_config_t::enableSrcMinorOffset
- (2) bool edma_minor_offset_config_t::enableDestMinorOffset
- (3) uint32_t edma_minor_offset_config_t::minorOffset

12.3.5 struct edma_tcd_t

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

Data Fields

- __IO uint32_t SADDR
SADDR register, used to save source address.
- __IO uint16_t SOFF
SOFF register, save offset bytes every transfer.
- __IO uint16_t ATTR

- **ATTR register; source/destination transfer size and modulo.**
- **_IO uint32_t NBYTES**
Nbytes register, minor loop length in bytes.
- **_IO uint32_t SLAST**
SLAST register.
- **_IO uint32_t DADDR**
DADDR register, used for destination address.
- **_IO uint16_t DOFF**
DOFF register, used for destination offset.
- **_IO uint16_t CITER**
CITER register, current minor loop numbers, for unfinished minor loop.
- **_IO uint32_t DLAST_SGA**
DLASTSGA register, next tcd address used in scatter-gather mode.
- **_IO uint16_t CSR**
CSR register, for TCD control status.
- **_IO uint16_t BITER**
BITER register, begin minor loop count.

Field Documentation

(1) **_IO uint16_t edma_tcd_t::CITER**

(2) **_IO uint16_t edma_tcd_t::BITER**

12.3.6 struct edma_handle_t

Data Fields

- **edma_callback callback**
Callback function for major count exhausted.
- **void * userData**
Callback function parameter.
- **DMA_Type * base**
eDMA peripheral base address.
- **edma_tcd_t * tcdPool**
Pointer to memory stored TCDs.
- **uint8_t channel**
eDMA channel number.
- **volatile int8_t header**
The first TCD index.
- **volatile int8_t tail**
The last TCD index.
- **volatile int8_t tcdUsed**
The number of used TCD slots.
- **volatile int8_t tcdSize**
The total number of TCD slots in the queue.
- **uint8_t flags**
The status of the current channel.

Field Documentation

- (1) `edma_callback edma_handle_t::callback`
- (2) `void* edma_handle_t::userData`
- (3) `DMA_Type* edma_handle_t::base`
- (4) `edma_tcd_t* edma_handle_t::tcdPool`
- (5) `uint8_t edma_handle_t::channel`
- (6) `volatile int8_t edma_handle_t::header`

Should point to the next TCD to be loaded into the eDMA engine.

- (7) `volatile int8_t edma_handle_t::tail`

Should point to the next TCD to be stored into the memory pool.

- (8) `volatile int8_t edma_handle_t::tcdUsed`

Should reflect the number of TCDs can be used/loaded in the memory.

- (9) `volatile int8_t edma_handle_t::tcdSize`

- (10) `uint8_t edma_handle_t::flags`

12.4 Macro Definition Documentation

12.4.1 #define FSL_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 3))

Version 2.4.3.

12.5 Typedef Documentation

12.5.1 `typedef void(* edma_callback)(struct _edma_handle *handle, void *userData, bool transferDone, uint32_t tcds)`

This callback function is called in the EDMA interrupt handle. In normal mode, run into callback function means the transfer users need is done. In scatter gather mode, run into callback function means a transfer control block (tcd) is finished. Not all transfer finished, users can get the finished tcd numbers using interface EDMA_GetUnusedTCDNumber.

Parameters

<i>handle</i>	EDMA handle pointer, users shall not touch the values inside.
<i>userData</i>	The callback user parameter pointer. Users can use this parameter to involve things users need to change in EDMA callback function.
<i>transferDone</i>	If the current loaded transfer done. In normal mode it means if all transfer done. In scatter gather mode, this parameter shows is the current transfer block in EDM-A register is done. As the load of core is different, it will be different if the new tcd loaded into EDMA registers while this callback called. If true, it always means new tcd still not loaded into registers, while false means new tcd already loaded into registers.
<i>tcds</i>	How many tcds are done from the last callback. This parameter only used in scatter gather mode. It tells user how many tcds are finished between the last callback and this.

12.6 Enumeration Type Documentation

12.6.1 enum edma_transfer_size_t

Enumerator

- kEDMA_TransferSize1Bytes*** Source/Destination data transfer size is 1 byte every time.
- kEDMA_TransferSize2Bytes*** Source/Destination data transfer size is 2 bytes every time.
- kEDMA_TransferSize4Bytes*** Source/Destination data transfer size is 4 bytes every time.
- kEDMA_TransferSize8Bytes*** Source/Destination data transfer size is 8 bytes every time.
- kEDMA_TransferSize16Bytes*** Source/Destination data transfer size is 16 bytes every time.
- kEDMA_TransferSize32Bytes*** Source/Destination data transfer size is 32 bytes every time.

12.6.2 enum edma_modulo_t

Enumerator

- kEDMA_ModuloDisable*** Disable modulo.
- kEDMA_Modulo2bytes*** Circular buffer size is 2 bytes.
- kEDMA_Modulo4bytes*** Circular buffer size is 4 bytes.
- kEDMA_Modulo8bytes*** Circular buffer size is 8 bytes.
- kEDMA_Modulo16bytes*** Circular buffer size is 16 bytes.
- kEDMA_Modulo32bytes*** Circular buffer size is 32 bytes.
- kEDMA_Modulo64bytes*** Circular buffer size is 64 bytes.
- kEDMA_Modulo128bytes*** Circular buffer size is 128 bytes.
- kEDMA_Modulo256bytes*** Circular buffer size is 256 bytes.
- kEDMA_Modulo512bytes*** Circular buffer size is 512 bytes.
- kEDMA_Modulo1Kbytes*** Circular buffer size is 1 K bytes.
- kEDMA_Modulo2Kbytes*** Circular buffer size is 2 K bytes.
- kEDMA_Modulo4Kbytes*** Circular buffer size is 4 K bytes.

kEDMA_Modulo8Kbytes Circular buffer size is 8 K bytes.
kEDMA_Modulo16Kbytes Circular buffer size is 16 K bytes.
kEDMA_Modulo32Kbytes Circular buffer size is 32 K bytes.
kEDMA_Modulo64Kbytes Circular buffer size is 64 K bytes.
kEDMA_Modulo128Kbytes Circular buffer size is 128 K bytes.
kEDMA_Modulo256Kbytes Circular buffer size is 256 K bytes.
kEDMA_Modulo512Kbytes Circular buffer size is 512 K bytes.
kEDMA_Modulo1Mbytes Circular buffer size is 1 M bytes.
kEDMA_Modulo2Mbytes Circular buffer size is 2 M bytes.
kEDMA_Modulo4Mbytes Circular buffer size is 4 M bytes.
kEDMA_Modulo8Mbytes Circular buffer size is 8 M bytes.
kEDMA_Modulo16Mbytes Circular buffer size is 16 M bytes.
kEDMA_Modulo32Mbytes Circular buffer size is 32 M bytes.
kEDMA_Modulo64Mbytes Circular buffer size is 64 M bytes.
kEDMA_Modulo128Mbytes Circular buffer size is 128 M bytes.
kEDMA_Modulo256Mbytes Circular buffer size is 256 M bytes.
kEDMA_Modulo512Mbytes Circular buffer size is 512 M bytes.
kEDMA_Modulo1Gbytes Circular buffer size is 1 G bytes.
kEDMA_Modulo2Gbytes Circular buffer size is 2 G bytes.

12.6.3 enum edma_bandwidth_t

Enumerator

kEDMA_BandwidthStallNone No eDMA engine stalls.
kEDMA_BandwidthStall4Cycle eDMA engine stalls for 4 cycles after each read/write.
kEDMA_BandwidthStall8Cycle eDMA engine stalls for 8 cycles after each read/write.

12.6.4 enum edma_channel_link_type_t

Enumerator

kEDMA_LinkNone No channel link.
kEDMA_MinorLink Channel link after each minor loop.
kEDMA_MajorLink Channel link while major loop count exhausted.

12.6.5 anonymous enum

Enumerator

kEDMA_DoneFlag DONE flag, set while transfer finished, CITER value exhausted.
kEDMA_ErrorFlag eDMA error flag, an error occurred in a transfer
kEDMA_InterruptFlag eDMA interrupt flag, set while an interrupt occurred of this channel

12.6.6 anonymous enum

Enumerator

- kEDMA_DestinationBusErrorFlag* Bus error on destination address.
- kEDMA_SourceBusErrorFlag* Bus error on the source address.
- kEDMA_ScatterGatherErrorFlag* Error on the Scatter/Gather address, not 32byte aligned.
- kEDMA_NbytesErrorFlag* NBYTES/CITER configuration error.
- kEDMA_DestinationOffsetErrorFlag* Destination offset not aligned with destination size.
- kEDMA_DestinationAddressErrorFlag* Destination address not aligned with destination size.
- kEDMA_SourceOffsetErrorFlag* Source offset not aligned with source size.
- kEDMA_SourceAddressErrorFlag* Source address not aligned with source size.
- kEDMA_ErrorChannelFlag* Error channel number of the cancelled channel number.
- kEDMA_ChannelPriorityErrorFlag* Channel priority is not unique.
- kEDMA_TransferCanceledFlag* Transfer cancelled.
- kEDMA_ValidFlag* No error occurred, this bit is 0. Otherwise, it is 1.

12.6.7 enum edma_interrupt_enable_t

Enumerator

- kEDMA_ErrorInterruptEnable* Enable interrupt while channel error occurs.
- kEDMA_MajorInterruptEnable* Enable interrupt while major count exhausted.
- kEDMA_HalfInterruptEnable* Enable interrupt while major count to half value.

12.6.8 enum edma_transfer_type_t

Enumerator

- kEDMA_MemoryToMemory* Transfer from memory to memory.
- kEDMA_PeripheralToMemory* Transfer from peripheral to memory.
- kEDMA_MemoryToPeripheral* Transfer from memory to peripheral.
- kEDMA_PeripheralToPeripheral* Transfer from Peripheral to peripheral.

12.6.9 anonymous enum

Enumerator

- kStatus_EDMA_QueueFull* TCD queue is full.
- kStatus_EDMA_Busy* Channel is busy and can't handle the transfer request.

12.7 Function Documentation

12.7.1 void EDMA_Init (DMA_Type * *base*, const edma_config_t * *config*)

This function ungates the eDMA clock and configures the eDMA peripheral according to the configuration structure.

Parameters

<i>base</i>	eDMA peripheral base address.
<i>config</i>	A pointer to the configuration structure, see "edma_config_t".

Note

This function enables the minor loop map feature.

12.7.2 void EDMA_Deinit (DMA_Type * *base*)

This function gates the eDMA clock.

Parameters

<i>base</i>	eDMA peripheral base address.
-------------	-------------------------------

12.7.3 void EDMA_InstallTCD (DMA_Type * *base*, uint32_t *channel*, edma_tcd_t * *tcd*)

Parameters

<i>base</i>	EDMA peripheral base address.
<i>channel</i>	EDMA channel number.
<i>tcd</i>	Point to TCD structure.

12.7.4 void EDMA_GetDefaultConfig (edma_config_t * *config*)

This function sets the configuration structure to default values. The default configuration is set to the following values.

```
* config.enableContinuousLinkMode = false;
* config.enableHaltOnError = true;
* config.enableRoundRobinArbitration = false;
* config.enableDebugMode = false;
*
```

Parameters

<i>config</i>	A pointer to the eDMA configuration structure.
---------------	--

12.7.5 static void EDMA_EnableContinuousChannelLinkMode (DMA_Type * *base*, bool *enable*) [inline], [static]

Note

Do not use continuous link mode with a channel linking to itself if there is only one minor loop iteration per service request, for example, if the channel's NBYTES value is the same as either the source or destination size. The same data transfer profile can be achieved by simply increasing the NBYTES value, which provides more efficient, faster processing.

Parameters

<i>base</i>	EDMA peripheral base address.
<i>enable</i>	true is enable, false is disable.

12.7.6 static void EDMA_EnableMinorLoopMapping (DMA_Type * *base*, bool *enable*) [inline], [static]

The TCDn.word2 is redefined to include individual enable fields, an offset field, and the NBYTES field.

Parameters

<i>base</i>	EDMA peripheral base address.
<i>enable</i>	true is enable, false is disable.

12.7.7 void EDMA_ResetChannel (DMA_Type * *base*, uint32_t *channel*)

This function sets TCD registers for this channel to default values.

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.

Note

This function must not be called while the channel transfer is ongoing or it causes unpredictable results.

This function enables the auto stop request feature.

12.7.8 void EDMA_SetTransferConfig (DMA_Type * *base*, uint32_t *channel*, const edma_transfer_config_t * *config*, edma_tcd_t * *nextTcd*)

This function configures the transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the TCD address. Example:

```
* edma_transfer_t config;
* edma_tcd_t tcd;
* config.srcAddr = ...;
* config.destAddr = ...;
* ...
* EDMA_SetTransferConfig(DMA0, channel, &config, &tcd);
*
```

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>config</i>	Pointer to eDMA transfer configuration structure.
<i>nextTcd</i>	Point to TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

Note

If nextTcd is not NULL, it means scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the eDMA_ResetChannel.

12.7.9 void EDMA_SetMinorOffsetConfig (DMA_Type * *base*, uint32_t *channel*, const edma_minor_offset_config_t * *config*)

The minor offset means that the signed-extended value is added to the source address or destination address after each minor loop.

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>config</i>	A pointer to the minor offset configuration structure.

12.7.10 void EDMA_SetChannelPreemptionConfig (DMA_Type * *base*, uint32_t *channel*, const edma_channel_Preemption_config_t * *config*)

This function configures the channel preemption attribute and the priority of the channel.

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number
<i>config</i>	A pointer to the channel preemption configuration structure.

12.7.11 void EDMA_SetChannelLink (DMA_Type * *base*, uint32_t *channel*, edma_channel_link_type_t *linkType*, uint32_t *linkedChannel*)

This function configures either the minor link or the major link mode. The minor link means that the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>linkType</i>	A channel link type, which can be one of the following: <ul style="list-style-type: none">• kEDMA_LinkNone• kEDMA_MinorLink• kEDMA_MajorLink

<i>linkedChannel</i>	The linked channel number.
----------------------	----------------------------

Note

Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

12.7.12 void EDMA_SetBandWidth (DMA_Type * *base*, uint32_t *channel*, edma_bandwidth_t *bandWidth*)

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>bandWidth</i>	A bandwidth setting, which can be one of the following: <ul style="list-style-type: none"> • kEDMABandwidthStallNone • kEDMABandwidthStall4Cycle • kEDMABandwidthStall8Cycle

12.7.13 void EDMA_SetModulo (DMA_Type * *base*, uint32_t *channel*, edma_modulo_t *srcModulo*, edma_modulo_t *destModulo*)

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.

<i>srcModulo</i>	A source modulo value.
<i>destModulo</i>	A destination modulo value.

12.7.14 static void EDMA_EnableAsyncRequest (DMA_Type * *base*, uint32_t *channel*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>enable</i>	The command to enable (true) or disable (false).

12.7.15 static void EDMA_EnableAutoStopRequest (DMA_Type * *base*, uint32_t *channel*, bool *enable*) [inline], [static]

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>enable</i>	The command to enable (true) or disable (false).

12.7.16 void EDMA_EnableChannelInterrupts (DMA_Type * *base*, uint32_t *channel*, uint32_t *mask*)

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>mask</i>	The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type.

12.7.17 void EDMA_DisableChannelInterrupts (DMA_Type * *base*, uint32_t *channel*, uint32_t *mask*)

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>mask</i>	The mask of the interrupt source to be set. Use the defined edma_interrupt_enable_t type.

12.7.18 void EDMA_SetMajorOffsetConfig (DMA_Type * *base*, uint32_t *channel*, int32_t *sourceOffset*, int32_t *destOffset*)

Adjustment value added to the source address at the completion of the major iteration count

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	edma channel number.
<i>sourceOffset</i>	source address offset will be applied to source address after major loop done.
<i>destOffset</i>	destination address offset will be applied to source address after major loop done.

12.7.19 void EDMA_TcdReset (edma_tcd_t * *tcd*)

This function sets all fields for this TCD structure to default value.

Parameters

<i>tcd</i>	Pointer to the TCD structure.
------------	-------------------------------

Note

This function enables the auto stop request feature.

12.7.20 void EDMA_TcdSetTransferConfig (edma_tcd_t * *tcd*, const edma_transfer_config_t * *config*, edma_tcd_t * *nextTcd*)

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The TCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```

*   edma_transfer_t config = {
*     ...
*   }
*   edma_tcd_t tcd __aligned(32);
*   edma_tcd_t nextTcd __aligned(32);
*   EDMA_TcdSetTransferConfig(&tcd, &config, &nextTcd);
*

```

Parameters

<i>tcd</i>	Pointer to the TCD structure.
<i>config</i>	Pointer to eDMA transfer configuration structure.
<i>nextTcd</i>	Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

Note

TCD address should be 32 bytes aligned or it causes an eDMA error.

If the nextTcd is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA_TcdReset.

12.7.21 void EDMA_TcdSetMinorOffsetConfig (*edma_tcd_t * tcd, const edma_minor_offset_config_t * config*)

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

Parameters

<i>tcd</i>	A point to the TCD structure.
<i>config</i>	A pointer to the minor offset configuration structure.

12.7.22 void EDMA_TcdSetChannelLink (*edma_tcd_t * tcd, edma_channel_link_type_t linkType, uint32_t linkedChannel*)

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note

Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

<i>tcd</i>	Point to the TCD structure.
<i>linkType</i>	Channel link type, it can be one of: <ul style="list-style-type: none"> • kEDMA_LinkNone • kEDMA_MinorLink • kEDMA_MajorLink
<i>linkedChannel</i>	The linked channel number.

12.7.23 static void EDMA_TcdSetBandWidth (*edma_tcd_t * tcd*, *edma_bandwidth_t bandWidth*) [inline], [static]

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

<i>tcd</i>	A pointer to the TCD structure.
<i>bandWidth</i>	A bandwidth setting, which can be one of the following: <ul style="list-style-type: none"> • kEDMABandwidthStallNone • kEDMABandwidthStall4Cycle • kEDMABandwidthStall8Cycle

12.7.24 void EDMA_TcdSetModulo (*edma_tcd_t * tcd*, *edma_modulo_t srcModulo*, *edma_modulo_t destModulo*)

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

<i>tcd</i>	A pointer to the TCD structure.
------------	---------------------------------

<i>srcModulo</i>	A source modulo value.
<i>destModulo</i>	A destination modulo value.

12.7.25 static void EDMA_TcdEnableAutoStopRequest (*edma_tcd_t * tcd, bool enable*) [inline], [static]

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

<i>tcd</i>	A pointer to the TCD structure.
<i>enable</i>	The command to enable (true) or disable (false).

12.7.26 void EDMA_TcdEnableInterrupts (*edma_tcd_t * tcd, uint32_t mask*)

Parameters

<i>tcd</i>	Point to the TCD structure.
<i>mask</i>	The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type.

12.7.27 void EDMA_TcdDisableInterrupts (*edma_tcd_t * tcd, uint32_t mask*)

Parameters

<i>tcd</i>	Point to the TCD structure.
<i>mask</i>	The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type.

12.7.28 void EDMA_TcdSetMajorOffsetConfig (*edma_tcd_t * tcd, int32_t sourceOffset, int32_t destOffset*)

Adjustment value added to the source address at the completion of the major iteration count

Parameters

<i>tcd</i>	A point to the TCD structure.
<i>sourceOffset</i>	source address offset will be applied to source address after major loop done.
<i>destOffset</i>	destination address offset will be applied to source address after major loop done.

12.7.29 static void EDMA_EnableChannelRequest (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

This function enables the hardware channel request.

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.

12.7.30 static void EDMA_DisableChannelRequest (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

This function disables the hardware channel request.

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.

12.7.31 static void EDMA_TriggerChannelStart (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

This function starts a minor loop transfer.

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.

12.7.32 **uint32_t EDMA_GetRemainingMajorLoopCount (DMA_Type * *base*, uint32_t *channel*)**

This function checks the TCD (Task Control Descriptor) status for a specified eDMA channel and returns the number of major loop count that has not finished.

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.

Returns

Major loop count which has not been transferred yet for the current TCD.

Note

1. This function can only be used to get unfinished major loop count of transfer without the next TCD, or it might be inaccuracy.
 1. The unfinished/remaining transfer bytes cannot be obtained directly from registers while the channel is running. Because to calculate the remaining bytes, the initial NBYTES configured in DMA_TCDn_NBYTES_MLNO register is needed while the eDMA IP does not support getting it while a channel is active. In another word, the NBYTES value reading is always the actual (decrementing) NBYTES value the dma_engine is working with while a channel is running. Consequently, to get the remaining transfer bytes, a software-saved initial value of NBYTES (for example copied before enabling the channel) is needed. The formula to calculate it is shown below: RemainingBytes = RemainingMajorLoopCount * NBYTE-S(initially configured)

12.7.33 static uint32_t EDMA_GetErrorStatusFlags (DMA_Type * *base*) [inline], [static]

Parameters

<i>base</i>	eDMA peripheral base address.
-------------	-------------------------------

Returns

The mask of error status flags. Users need to use the _edma_error_status_flags type to decode the return variables.

12.7.34 uint32_t EDMA_GetChannelStatusFlags (DMA_Type * *base*, uint32_t *channel*)

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.

Returns

The mask of channel status flags. Users need to use the `_edma_channel_status_flags` type to decode the return variables.

12.7.35 void EDMA_ClearChannelStatusFlags (DMA_Type * *base*, uint32_t *channel*, uint32_t *mask*)

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>mask</i>	The mask of channel status to be cleared. Users need to use the defined <code>_edma_channel_status_flags</code> type.

12.7.36 void EDMA_CreateHandle (edma_handle_t * *handle*, DMA_Type * *base*, uint32_t *channel*)

This function is called if using the transactional API for eDMA. This function initializes the internal state of the eDMA handle.

Parameters

<i>handle</i>	eDMA handle pointer. The eDMA handle stores callback function and parameters.
<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.

12.7.37 void EDMA_InstallTCDMemory (edma_handle_t * *handle*, edma_tcd_t * *tcdPool*, uint32_t *tcdSize*)

This function is called after the EDMA_CreateHandle to use scatter/gather feature. This function shall only be used while users need to use scatter gather mode. Scatter gather mode enables EDMA to load a new transfer control block (tcd) in hardware, and automatically reconfigure that DMA channel for a

new transfer. Users need to prepare tcd memory and also configure tcds using interface EDMA_SubmitTransfer.

Parameters

<i>handle</i>	eDMA handle pointer.
<i>tcdPool</i>	A memory pool to store TCDs. It must be 32 bytes aligned.
<i>tcdSize</i>	The number of TCD slots.

12.7.38 void EDMA_SetCallback (*edma_handle_t * handle, edma_callback callback, void * userData*)

This callback is called in the eDMA IRQ handler. Use the callback to do something after the current major loop transfer completes. This function will be called every time one tcd finished transfer.

Parameters

<i>handle</i>	eDMA handle pointer.
<i>callback</i>	eDMA callback function pointer.
<i>userData</i>	A parameter for the callback function.

12.7.39 void EDMA_PrepTransferConfig (*edma_transfer_config_t * config, void * srcAddr, uint32_t srcWidth, int16_t srcOffset, void * destAddr, uint32_t destWidth, int16_t destOffset, uint32_t bytesEachRequest, uint32_t transferBytes*)

This function prepares the transfer configuration structure according to the user input.

Parameters

<i>config</i>	The user configuration structure of type <i>edma_transfer_t</i> .
<i>srcAddr</i>	eDMA transfer source address.
<i>srcWidth</i>	eDMA transfer source address width(bytes).
<i>srcOffset</i>	source address offset.
<i>destAddr</i>	eDMA transfer destination address.
<i>destWidth</i>	eDMA transfer destination address width(bytes).
<i>destOffset</i>	destination address offset.
<i>bytesEachRequest</i>	eDMA transfer bytes per channel request.
<i>transferBytes</i>	eDMA transfer bytes to be transferred.

Note

The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

12.7.40 void EDMA_PrepTransfer (*edma_transfer_config_t * config, void * srcAddr, uint32_t srcWidth, void * destAddr, uint32_t destWidth, uint32_t bytesEachRequest, uint32_t transferBytes, edma_transfer_type_t transferType*)

This function prepares the transfer configuration structure according to the user input.

Parameters

<i>config</i>	The user configuration structure of type <code>edma_transfer_t</code> .
<i>srcAddr</i>	eDMA transfer source address.
<i>srcWidth</i>	eDMA transfer source address width(bytes).
<i>destAddr</i>	eDMA transfer destination address.
<i>destWidth</i>	eDMA transfer destination address width(bytes).
<i>bytesEachRequest</i>	eDMA transfer bytes per channel request.
<i>transferBytes</i>	eDMA transfer bytes to be transferred.
<i>transferType</i>	eDMA transfer type.

Note

The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

12.7.41 status_t EDMA_SubmitTransfer (*edma_handle_t * handle, const edma_transfer_config_t * config*)

This function submits the eDMA transfer request according to the transfer configuration structure. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function `EDMA_InstallTCDMemory` before.

Parameters

<i>handle</i>	eDMA handle pointer.
<i>config</i>	Pointer to eDMA transfer configuration structure.

Return values

<i>kStatus_EDMA_Success</i>	It means submit transfer request succeed.
<i>kStatus_EDMA_Queue-Full</i>	It means TCD queue is full. Submit transfer request is not allowed.
<i>kStatus_EDMA_Busy</i>	It means the given channel is busy, need to submit request later.

12.7.42 void EDMA_StartTransfer (*edma_handle_t * handle*)

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

Parameters

<i>handle</i>	eDMA handle pointer.
---------------	----------------------

12.7.43 void EDMA_StopTransfer (*edma_handle_t * handle*)

This function disables the channel request to pause the transfer. Users can call [EDMA_StartTransfer\(\)](#) again to resume the transfer.

Parameters

<i>handle</i>	eDMA handle pointer.
---------------	----------------------

12.7.44 void EDMA_AbortTransfer (*edma_handle_t * handle*)

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

Parameters

<i>handle</i>	DMA handle pointer.
---------------	---------------------

12.7.45 static uint32_t EDMA_GetUnusedTCDNumber (edma_handle_t * *handle*) [inline], [static]

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.

Parameters

<i>handle</i>	DMA handle pointer.
---------------	---------------------

Returns

The unused tcd slot number.

12.7.46 static uint32_t EDMA_GetNextTCDAccount (edma_handle_t * *handle*) [inline], [static]

This function gets the next tcd address. If this is last TCD, return 0.

Parameters

<i>handle</i>	DMA handle pointer.
---------------	---------------------

Returns

The next TCD address.

12.7.47 void EDMA_HandleIRQ (edma_handle_t * *handle*)

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Note: For the case using TCD queue, when the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor (if scatter/gather is enabled).

For instance, when the time interrupt of TCD[0] happens, the TCD[1] has already been loaded into the eDMA engine. As sga and sga_index are calculated based on the DLAST_SGA bitfield lies in the TCD_CSR register, the sga_index in this case should be 2 (DLAST_SGA of TCD[1] stores the address of TCD[2]). Thus, the "tcdUsed" updated should be (tcdUsed - 2U) which indicates the number of TCDs can be loaded in the memory pool (because TCD[0] and TCD[1] have been loaded into the eDMA engine at this point already.).

For the last two continuous ISRs in a scatter/gather process, they both load the last TCD (The last ISR does not load a new TCD) from the memory pool to the eDMA engine when major loop completes. Therefore, ensure that the header and tcdUsed updated are identical for them. tcdUsed are both 0 in this case as no TCD to be loaded.

See the "eDMA basic data flow" in the eDMA Functional description section of the Reference Manual for further details.

Parameters

<i>handle</i>	eDMA handle pointer.
---------------	----------------------

Chapter 13

EWM: External Watchdog Monitor Driver

13.1 Overview

The MCUXpresso SDK provides a peripheral driver for the External Watchdog (EWM) Driver module of MCUXpresso SDK devices.

13.2 Typical use case

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

Data Structures

- struct `ewm_config_t`
Describes EWM clock source. [More...](#)

Enumerations

- enum `_ewm_interrupt_enable_t` { `kEWM_InterruptEnable` = EWM_CTRL_INTEN_MASK }
EWM interrupt configuration structure with default settings all disabled.
- enum `_ewm_status_flags_t` { `kEWM_RunningFlag` = EWM_CTRL_EWMEN_MASK }
EWM status flags.

Driver version

- #define `FSL_EWM_DRIVER_VERSION` (MAKE_VERSION(2, 0, 3))
EWM driver version 2.0.3.

EWM initialization and de-initialization

- void `EWM_Init` (EWM_Type *base, const `ewm_config_t` *config)
Initializes the EWM peripheral.
- void `EWM_Deinit` (EWM_Type *base)
Deinitializes the EWM peripheral.
- void `EWM_GetDefaultConfig` (`ewm_config_t` *config)
Initializes the EWM configuration structure.

EWM functional Operation

- static void `EWM_EnableInterrupts` (EWM_Type *base, uint32_t mask)
Enables the EWM interrupt.
- static void `EWM_DisableInterrupts` (EWM_Type *base, uint32_t mask)
Disables the EWM interrupt.
- static uint32_t `EWM_GetStatusFlags` (EWM_Type *base)
Gets all status flags.

- void **EWM_Refresh** (EWM_Type *base)
Services the EWM.

13.3 Data Structure Documentation

13.3.1 struct ewm_config_t

Data structure for EWM configuration.

This structure is used to configure the EWM.

Data Fields

- bool **enableEwm**
Enable EWM module.
- bool **enableEwmInput**
Enable EWM_in input.
- bool **setInputAssertLogic**
EWM_in signal assertion state.
- bool **enableInterrupt**
Enable EWM interrupt.
- uint8_t **prescaler**
Clock prescaler value.
- uint8_t **compareLowValue**
Compare low-register value.
- uint8_t **compareHighValue**
Compare high-register value.

13.4 Macro Definition Documentation

13.4.1 #define FSL_EWM_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

13.5 Enumeration Type Documentation

13.5.1 enum _ewm_interrupt_enable_t

This structure contains the settings for all of EWM interrupt configurations.

Enumerator

kEWM InterruptEnable Enable the EWM to generate an interrupt.

13.5.2 enum _ewm_status_flags_t

This structure contains the constants for the EWM status flags for use in the EWM functions.

Enumerator

kEWM_RunningFlag Running flag, set when EWM is enabled.

13.6 Function Documentation

13.6.1 void EWM_Init (EWM_Type * *base*, const ewm_config_t * *config*)

This function is used to initialize the EWM. After calling, the EWM runs immediately according to the configuration. Note that, except for the interrupt enable control bit, other control bits and registers are write once after a CPU reset. Modifying them more than once generates a bus transfer error.

This is an example.

```
*     ewm_config_t config;
*     EWM_GetDefaultConfig(&config);
*     config.compareHighValue = 0xAAU;
*     EWM_Init(ewm_base, &config);
*
```

Parameters

<i>base</i>	EWM peripheral base address
<i>config</i>	The configuration of the EWM

13.6.2 void EWM_Deinit (EWM_Type * *base*)

This function is used to shut down the EWM.

Parameters

<i>base</i>	EWM peripheral base address
-------------	-----------------------------

13.6.3 void EWM_GetDefaultConfig (ewm_config_t * *config*)

This function initializes the EWM configuration structure to default values. The default values are as follows.

```
*     ewmConfig->enableEwm = true;
*     ewmConfig->enableEwmInput = false;
*     ewmConfig->setInputAssertLogic = false;
*     ewmConfig->enableInterrupt = false;
*     ewmConfig->ewm_lpo_clock_source_t = kEWM_LpoClockSource0;
*     ewmConfig->prescaler = 0;
*     ewmConfig->compareLowValue = 0;
*     ewmConfig->compareHighValue = 0xFEU;
*
```

Parameters

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

See Also

[ewm_config_t](#)

13.6.4 static void EWM_EnableInterrupts (**EWM_Type** * *base*, **uint32_t** *mask*) [**inline**], [**static**]

This function enables the EWM interrupt.

Parameters

<i>base</i>	EWM peripheral base address
<i>mask</i>	The interrupts to enable The parameter can be combination of the following source if defined <ul style="list-style-type: none">• kEWM InterruptEnable

13.6.5 static void EWM_DisableInterrupts (**EWM_Type** * *base*, **uint32_t** *mask*) [**inline**], [**static**]

This function enables the EWM interrupt.

Parameters

<i>base</i>	EWM peripheral base address
<i>mask</i>	The interrupts to disable The parameter can be combination of the following source if defined <ul style="list-style-type: none">• kEWM InterruptEnable

13.6.6 static **uint32_t** EWM_GetStatusFlags (**EWM_Type** * *base*) [**inline**], [**static**]

This function gets all status flags.

This is an example for getting the running flag.

```
*     uint32_t status;
*     status = EWM_GetStatusFlags(ewm_base) & kEWM_RunningFlag;
*
```

Parameters

<i>base</i>	EWM peripheral base address
-------------	-----------------------------

Returns

State of the status flag: asserted (true) or not-asserted (false).

See Also

[_ewm_status_flags_t](#)

- True: a related status flag has been set.
- False: a related status flag is not set.

13.6.7 void EWM_Refresh (EWM_Type * *base*)

This function resets the EWM counter to zero.

Parameters

<i>base</i>	EWM peripheral base address
-------------	-----------------------------

Chapter 14

C90TFS Flash Driver

14.1 Overview

The flash provides the C90TFS Flash driver of Kinetis devices with the C90TFS Flash module inside. The flash driver provides general APIs to handle specific operations on C90TFS/FTFx Flash module. The user can use those APIs directly in the application. In addition, it provides internal functions called by the driver. Although these functions are not meant to be called from the user's application directly, the APIs can still be used.

Modules

- [Ftftx CACHE Driver](#)
- [Ftftx FLASH Driver](#)
- [Ftftx FLEXNVM Driver](#)
- [ftfx controller](#)
- [ftfx feature](#)

14.2 Ftftx FLASH Driver

14.2.1 Overview

Data Structures

- union `pflash_prot_status_t`
PFlash protection status. [More...](#)
- struct `flash_config_t`
Flash driver state information. [More...](#)

Enumerations

- enum `flash_prot_state_t` {
 `kFLASH_ProtectionStateUnprotected`,
 `kFLASH_ProtectionStateProtected`,
 `kFLASH_ProtectionStateMixed` }

Enumeration for the three possible flash protection levels.
- enum `flash_xacc_state_t` {
 `kFLASH_AccessStateUnLimited`,
 `kFLASH_AccessStateExecuteOnly`,
 `kFLASH_AccessStateMixed` }

Enumeration for the three possible flash execute access levels.
- enum `flash_property_tag_t` {
 `kFLASH_PropertyPflash0SectorSize` = 0x00U,
 `kFLASH_PropertyPflash0TotalSize` = 0x01U,
 `kFLASH_PropertyPflash0BlockSize` = 0x02U,
 `kFLASH_PropertyPflash0BlockCount` = 0x03U,
 `kFLASH_PropertyPflash0BlockBaseAddr` = 0x04U,
 `kFLASH_PropertyPflash0FacSupport` = 0x05U,
 `kFLASH_PropertyPflash0AccessSegmentSize` = 0x06U,
 `kFLASH_PropertyPflash0AccessSegmentCount` = 0x07U,
 `kFLASH_PropertyPflash1SectorSize` = 0x10U,
 `kFLASH_PropertyPflash1TotalSize` = 0x11U,
 `kFLASH_PropertyPflash1BlockSize` = 0x12U,
 `kFLASH_PropertyPflash1BlockCount` = 0x13U,
 `kFLASH_PropertyPflash1BlockBaseAddr` = 0x14U,
 `kFLASH_PropertyPflash1FacSupport` = 0x15U,
 `kFLASH_PropertyPflash1AccessSegmentSize` = 0x16U,
 `kFLASH_PropertyPflash1AccessSegmentCount` = 0x17U,
 `kFLASH_PropertyFlexRamBlockBaseAddr` = 0x20U,
 `kFLASH_PropertyFlexRamTotalSize` = 0x21U }

Enumeration for various flash properties.

Flash version

- #define **FSL_FLASH_DRIVER_VERSION** (MAKE_VERSION(3U, 1U, 2U))
Flash driver version for SDK.
- #define **FSL_FLASH_DRIVER_VERSION_ROM** (MAKE_VERSION(3U, 0U, 0U))
Flash driver version for ROM.

Initialization

- **status_t FLASH_Init (flash_config_t *config)**
Initializes the global flash properties structure members.

Erasing

- **status_t FLASH_Erase (flash_config_t *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)**
Erases the Dflash sectors encompassed by parameters passed into function.
- **status_t FLASH_EraseSectorNonBlocking (flash_config_t *config, uint32_t start, uint32_t key)**
Erases the Dflash sectors encompassed by parameters passed into function.
- **status_t FLASH_EraseAll (flash_config_t *config, uint32_t key)**
Erases entire flexnvm.

Programming

- **status_t FLASH_Program (flash_config_t *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)**
Programs flash with data at locations passed in through parameters.
- **status_t FLASH_ProgramOnce (flash_config_t *config, uint32_t index, uint8_t *src, uint32_t lengthInBytes)**
Program the Program-Once-Field through parameters.

Reading

- **status_t FLASH_ReadResource (flash_config_t *config, uint32_t start, uint8_t *dst, uint32_t lengthInBytes, ftfx_read_resource_opt_t option)**
Reads the resource with data at locations passed in through parameters.
- **status_t FLASH_ReadOnce (flash_config_t *config, uint32_t index, uint8_t *dst, uint32_t lengthInBytes)**
Reads the Program Once Field through parameters.

Verification

- **status_t FLASH_VerifyErase (flash_config_t *config, uint32_t start, uint32_t lengthInBytes, ftfx_margin_value_t margin)**

- Verifies an erasure of the desired flash area at a specified margin level.
- `status_t FLASH_VerifyEraseAll (flash_config_t *config, ftfx_margin_value_t margin)`
Verifies erasure of the entire flash at a specified margin level.
- `status_t FLASH_VerifyProgram (flash_config_t *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, ftfx_margin_value_t margin, uint32_t *failedAddress, uint32_t *failedData)`
Verifies programming of the desired flash area at a specified margin level.

Security

- `status_t FLASH_GetSecurityState (flash_config_t *config, ftfx_security_state_t *state)`
Returns the security state via the pointer passed into the function.
- `status_t FLASH_SecurityBypass (flash_config_t *config, const uint8_t *backdoorKey)`
Allows users to bypass security with a backdoor key.

Protection

- `status_t FLASH_IsProtected (flash_config_t *config, uint32_t start, uint32_t lengthInBytes, flash_prot_state_t *protection_state)`
Returns the protection state of the desired flash area via the pointer passed into the function.
- `status_t FLASH_IsExecuteOnly (flash_config_t *config, uint32_t start, uint32_t lengthInBytes, flash_xacc_state_t *access_state)`
Returns the access state of the desired flash area via the pointer passed into the function.
- `status_t FLASH_PflashSetProtection (flash_config_t *config, pflash_prot_status_t *protectStatus)`
Sets the PFlash Protection to the intended protection status.
- `status_t FLASH_PflashGetProtection (flash_config_t *config, pflash_prot_status_t *protectStatus)`
Gets the PFlash protection status.

Properties

- `status_t FLASHGetProperty (flash_config_t *config, flash_property_tag_t whichProperty, uint32_t *value)`
Returns the desired flash property.

commandStatus

- `status_t FLASH_GetCommandState (void)`
Get previous command status.

14.2.2 Data Structure Documentation

14.2.2.1 union pflash_prot_status_t

Data Fields

- `uint32_t protl`
 $PROT[31:0]$.
- `uint32_t proth`
 $PROT[63:32]$.
- `uint8_t protsl`
 $PROTS[7:0]$.
- `uint8_t protsh`
 $PROTS[15:8]$.

Field Documentation

- (1) `uint32_t pflash_prot_status_t::protl`
- (2) `uint32_t pflash_prot_status_t::proth`
- (3) `uint8_t pflash_prot_status_t::protsl`
- (4) `uint8_t pflash_prot_status_t::protsh`

14.2.2.2 struct flash_config_t

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

14.2.3 Macro Definition Documentation

14.2.3.1 #define FSL_FLASH_DRIVER_VERSION (MAKE_VERSION(3U, 1U, 2U))

Version 3.1.2.

14.2.3.2 #define FSL_FLASH_DRIVER_VERSION_ROM (MAKE_VERSION(3U, 0U, 0U))

Version 3.0.0.

14.2.4 Enumeration Type Documentation

14.2.4.1 enum flash_prot_state_t

Enumerator

kFLASH_ProtectionStateUnprotected Flash region is not protected.

kFLASH_ProtectionStateProtected Flash region is protected.

kFLASH_ProtectionStateMixed Flash is mixed with protected and unprotected region.

14.2.4.2 enum flash_xacc_state_t

Enumerator

kFLASH_AccessStateUnLimited Flash region is unlimited.

kFLASH_AccessStateExecuteOnly Flash region is execute only.

kFLASH_AccessStateMixed Flash is mixed with unlimited and execute only region.

14.2.4.3 enum flash_property_tag_t

Enumerator

kFLASH_PropertyPflash0SectorSize Pflash sector size property.

kFLASH_PropertyPflash0TotalSize Pflash total size property.

kFLASH_PropertyPflash0BlockSize Pflash block size property.

kFLASH_PropertyPflash0BlockCount Pflash block count property.

kFLASH_PropertyPflash0BlockBaseAddr Pflash block base address property.

kFLASH_PropertyPflash0FacSupport Pflash fac support property.

kFLASH_PropertyPflash0AccessSegmentSize Pflash access segment size property.

kFLASH_PropertyPflash0AccessSegmentCount Pflash access segment count property.

kFLASH_PropertyPflash1SectorSize Pflash sector size property.

kFLASH_PropertyPflash1TotalSize Pflash total size property.

kFLASH_PropertyPflash1BlockSize Pflash block size property.

kFLASH_PropertyPflash1BlockCount Pflash block count property.

kFLASH_PropertyPflash1BlockBaseAddr Pflash block base address property.

kFLASH_PropertyPflash1FacSupport Pflash fac support property.

kFLASH_PropertyPflash1AccessSegmentSize Pflash access segment size property.

kFLASH_PropertyPflash1AccessSegmentCount Pflash access segment count property.

kFLASH_PropertyFlexRamBlockBaseAddr FlexRam block base address property.

kFLASH_PropertyFlexRamTotalSize FlexRam total size property.

14.2.5 Function Documentation

14.2.5.1 status_t FLASH_Init (flash_config_t * config)

This function checks and initializes the Flash module for the other Flash APIs.

Parameters

<i>config</i>	Pointer to the storage for the driver runtime state.
---------------	--

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_PartitionStatusUpdateFailure</i>	Failed to update the partition status.

14.2.5.2 status_t FLASH_Erase (flash_config_t * *config*, uint32_t *start*, uint32_t *lengthInBytes*, uint32_t *key*)

This function erases the appropriate number of flash sectors based on the desired start address and length.

Parameters

<i>config</i>	The pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words) to be erased. Must be word-aligned.
<i>key</i>	The value used to validate all flash erase APIs.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the appropriate number of flash sectors based on the desired start address and length were erased successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	The parameter is not aligned with the specified baseline.

<i>kStatus_FTFx_Address_Error</i>	The address is out of range.
<i>kStatus_FTFx_EraseKey_Error</i>	The API erase key is invalid.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access_Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_-CommandFailure</i>	Run-time error during the command execution.

14.2.5.3 **status_t FLASH_EraseSectorNonBlocking (flash_config_t * config, uint32_t start, uint32_t key)**

This function erases one flash sector size based on the start address, and it is executed asynchronously.

NOTE: This function can only erase one flash sector at a time, and the other commands can be executed after the previous command has been completed.

Parameters

<i>config</i>	The pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
<i>key</i>	The value used to validate all flash erase APIs.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_-AlignmentError</i>	The parameter is not aligned with the specified baseline.

<i>kStatus_FTFx_Address_Error</i>	The address is out of range.
<i>kStatus_FTFx_EraseKey_Error</i>	The API erase key is invalid.

14.2.5.4 status_t FLASH_EraseAll (**flash_config_t * config, uint32_t key**)

Parameters

<i>config</i>	Pointer to the storage for the driver runtime state.
<i>key</i>	A value used to validate all flash erase APIs.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the all pflash and flexnvm were erased successfully, the swap and eeprom have been reset to unconfigured state.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_EraseKey_Error</i>	API erase key is invalid.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during command execution.
<i>kStatus_FTFx_PartitionStatusUpdateFailure</i>	Failed to update the partition status.

14.2.5.5 status_t FLASH_Program (**flash_config_t * config, uint32_t start, uint8_t * src, uint32_t lengthInBytes**)

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be programmed. Must be word-aligned.
<i>src</i>	A pointer to the source buffer of data that is to be programmed into the flash.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the desired data were programmed successfully into flash based on desired start address and length.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

14.2.5.6 status_t FLASH_ProgramOnce (flash_config_t * config, uint32_t index, uint8_t * src, uint32_t lengthInBytes)

This function Program the Program-once-field with given index and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>index</i>	The index indicating the area of program once field to be read.
<i>src</i>	A pointer to the source buffer of data that is used to store data to be write.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; The index indicating the area of program once field was programmed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

14.2.5.7 **status_t FLASH_ReadResource(flash_config_t * config, uint32_t start, uint8_t * dst, uint32_t lengthInBytes, ftfx_read_resource_opt_t option)**

This function reads the flash memory with the desired location for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be programmed. Must be word-aligned.
<i>dst</i>	A pointer to the destination buffer of data that is used to store data to be read.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be read. Must be word-aligned.

<i>option</i>	The resource option which indicates which area should be read back.
---------------	---

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the data have been read successfully from program flash IFR, data flash IFR space, and the Version ID field.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

14.2.5.8 status_t FLASH_ReadOnce (*flash_config_t * config, uint32_t index, uint8_t * dst, uint32_t lengthInBytes*)

This function reads the read once feild with given index and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>index</i>	The index indicating the area of program once field to be read.
<i>dst</i>	A pointer to the destination buffer of data that is used to store data to be read.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the data have been successfully read form Program flash0 IFR map and Program Once field based on index and length.
-----------------------------	---

<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

14.2.5.9 **status_t FLASH_VerifyErase (flash_config_t * config, uint32_t start, uint32_t lengthInBytes, ftfx_margin_value_t margin)**

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
<i>margin</i>	Read margin choice.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the specified FLASH region has been erased.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with specified baseline.

<i>kStatus_FTFx_Address_Error</i>	Address is out of range.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

14.2.5.10 **status_t FLASH_VerifyEraseAll (flash_config_t * config, ftfx_margin_value_t margin)**

This function checks whether the flash is erased to the specified read margin level.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>margin</i>	Read margin choice.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; all program flash and flexnvm were in erased state.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.

<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.
------------------------------------	--

14.2.5.11 **status_t FLASH_VerifyProgram (flash_config_t * config, uint32_t start, uint32_t lengthInBytes, const uint8_t * expectedData, ftfx_margin_value_t margin, uint32_t * failedAddress, uint32_t * failedData)**

This function verifies the data programmed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be verified. Must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
<i>expectedData</i>	A pointer to the expected data that is to be verified against.
<i>margin</i>	Read margin choice.
<i>failedAddress</i>	A pointer to the returned failing address.
<i>failedData</i>	A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the desired data have been successfully programmed into specified FLASH region.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_-AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FTFx_Address-Error</i>	Address is out of range.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.

<i>kStatus_FTFx_Access_Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

14.2.5.12 status_t FLASH_GetSecurityState (**flash_config_t * config, ftfx_security_state_t * state**)

This function retrieves the current flash security status, including the security enabling state and the backdoor key enabling state.

Parameters

<i>config</i>	A pointer to storage for the driver runtime state.
<i>state</i>	A pointer to the value returned for the current security status code:

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the security state of flash was stored to state.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.

14.2.5.13 status_t FLASH_SecurityBypass (**flash_config_t * config, const uint8_t * backdoorKey**)

If the MCU is in secured state, this function unsecures the MCU by comparing the provided backdoor key with ones in the flash configuration field.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>backdoorKey</i>	A pointer to the user buffer containing the backdoor key.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

14.2.5.14 status_t FLASH_IsProtected (*flash_config_t * config*, *uint32_t start*, *uint32_t lengthInBytes*, *flash_prot_state_t * protection_state*)

This function retrieves the current flash protect status for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be checked. Must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words) to be checked. Must be word-aligned.
<i>protection_state</i>	A pointer to the value returned for the current protection status code for the desired flash area.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the protection state of specified FLASH region was stored to protection_state.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.

<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FTFx_AddressError</i>	The address is out of range.

14.2.5.15 status_t FLASH_IsExecuteOnly (*flash_config_t * config, uint32_t start, uint32_t lengthInBytes, flash_xacc_state_t * access_state*)

This function retrieves the current flash access status for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be checked. Must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be checked. Must be word-aligned.
<i>access_state</i>	A pointer to the value returned for the current access status code for the desired flash area.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the executeOnly state of specified FLASH region was stored to access_state.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	The parameter is not aligned to the specified baseline.
<i>kStatus_FTFx_AddressError</i>	The address is out of range.

14.2.5.16 status_t FLASH_PflashSetProtection (*flash_config_t * config, pflash_prot_status_t * protectStatus*)

Parameters

<i>config</i>	A pointer to storage for the driver runtime state.
<i>protectStatus</i>	The expected protect status to set to the PFlash protection register. Each bit is corresponding to protection of 1/32(64) of the total PFlash. The least significant bit is corresponding to the lowest address area of PFlash. The most significant bit is corresponding to the highest address area of PFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the specified FLASH region is protected.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx-CommandFailure</i>	Run-time error during command execution.

14.2.5.17 **status_t FLASH_PflashGetProtection (flash_config_t * *config*, pflash_prot_status_t * *protectStatus*)**

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>protectStatus</i>	Protect status returned by the PFlash IP. Each bit is corresponding to the protection of 1/32(64) of the total PFlash. The least significant bit corresponds to the lowest address area of the PFlash. The most significant bit corresponds to the highest address area of PFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the Protection state was stored to protect-Status;
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.

14.2.5.18 **status_t FLASH_GetProperty (flash_config_t * *config*, flash_property_tag_t *whichProperty*, uint32_t * *value*)**

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>whichProperty</i>	The desired property from the list of properties in enum flash_property_tag_t
<i>value</i>	A pointer to the value returned for the desired flash property.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the flash property was stored to value.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_UnknownProperty</i>	An unknown property tag.

14.2.5.19 status_t FLASH_GetCommandState (void)

This function is used to obtain the execution status of the previous command.

Return values

<i>kStatus_FTFx_Success</i>	The previous command is executed successfully.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx-CommandFailure</i>	Run-time error during the command execution.

14.3 Fftfx CACHE Driver

14.3.1 Overview

Data Structures

- struct `fftfx_prefetch_speculation_status_t`
FTFx prefetch speculation status. [More...](#)
- struct `fftfx_cache_config_t`
FTFx cache driver state information. [More...](#)

Enumerations

- enum `_fftfx_cache_ram_func_constants` { `kFTFx_CACHE_RamFuncMaxSizeInWords` = 16U }
Constants for execute-in-RAM flash function.

Functions

- `status_t FTFx_CACHE_Init (fftfx_cache_config_t *config)`
Initializes the global FTFx cache structure members.
- `status_t FTFx_CACHE_ClearCachePrefetchSpeculation (fftfx_cache_config_t *config, bool isPreProcess)`
Process the cache/prefetch/speculation to the flash.
- `status_t FTFx_CACHE_PflashSetPrefetchSpeculation (fftfx_prefetch_speculation_status_t *speculationStatus)`
Sets the PFlash prefetch speculation to the intended speculation status.
- `status_t FTFx_CACHE_PflashGetPrefetchSpeculation (fftfx_prefetch_speculation_status_t *speculationStatus)`
Gets the PFlash prefetch speculation status.

14.3.2 Data Structure Documentation

14.3.2.1 struct fftfx_prefetch_speculation_status_t

Data Fields

- `bool instructionOff`
Instruction speculation.
- `bool dataOff`
Data speculation.

Field Documentation

(1) `bool fftfx_prefetch_speculation_status_t::instructionOff`

(2) `bool ftfx_prefetch_speculation_status_t::dataOff`

14.3.2.2 struct ftfx_cache_config_t

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

Data Fields

- `uint8_t flashMemoryIndex`
0 - primary flash; 1 - secondary flash
- `function_bit_operation_ptr_t bitOperFuncAddr`
An buffer point to the flash execute-in-RAM function.

Field Documentation

(1) `function_bit_operation_ptr_t ftfx_cache_config_t::bitOperFuncAddr`

14.3.3 Enumeration Type Documentation

14.3.3.1 enum _ftfx_cache_ram_func_constants

Enumerator

`kFTFx_CACHE_RamFuncMaxSizeInWords` The maximum size of execute-in-RAM function.

14.3.4 Function Documentation

14.3.4.1 status_t FTFx_CACHE_Init (`ftfx_cache_config_t * config`)

This function checks and initializes the Flash module for the other FTFx cache APIs.

Parameters

<code>config</code>	Pointer to the storage for the driver runtime state.
---------------------	--

Return values

<code>kStatus_FTFx_Success</code>	API was executed successfully.
<code>kStatus_FTFx_Invalid-Argument</code>	An invalid argument is provided.

<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
---	---

14.3.4.2 status_t FTFx_CACHE_ClearCachePrefetchSpeculation (*ftfx_cache_config_t * config, bool isPreProcess*)

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>isPreProcess</i>	The possible option used to control flash cache/prefetch/speculation

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_Invalid-Argument</i>	Invalid argument is provided.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.

14.3.4.3 status_t FTFx_CACHE_PflashSetPrefetchSpeculation (*ftfx_prefetch_speculation_status_t * speculationStatus*)

Parameters

<i>speculation-Status</i>	The expected protect status to set to the PFlash protection register. Each bit is
---------------------------	---

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_Invalid-SpeculationOption</i>	An invalid speculation option argument is provided.

14.3.4.4 status_t FTFx_CACHE_PflashGetPrefetchSpeculation (*ftfx_prefetch_speculation_status_t * speculationStatus*)

Parameters

<i>speculation- Status</i>	Speculation status returned by the PFlash IP.
--------------------------------	---

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
-----------------------------	--------------------------------

14.4 Ftftx FLEXNVM Driver

14.4.1 Overview

Data Structures

- struct `flexnvm_config_t`
Flexnvm driver state information. [More...](#)

Enumerations

- enum `flexnvm_property_tag_t` {

`kFLEXNVM_PropertyDflashSectorSize` = 0x00U,
`kFLEXNVM_PropertyDflashTotalSize` = 0x01U,
`kFLEXNVM_PropertyDflashBlockSize` = 0x02U,
`kFLEXNVM_PropertyDflashBlockCount` = 0x03U,
`kFLEXNVM_PropertyDflashBlockBaseAddr` = 0x04U,
`kFLEXNVM_PropertyAliasDflashBlockBaseAddr` = 0x05U,
`kFLEXNVM_PropertyFlexRamBlockBaseAddr` = 0x06U,
`kFLEXNVM_PropertyFlexRamTotalSize` = 0x07U,
`kFLEXNVM_PropertyEepromTotalSize` = 0x08U }

Enumeration for various flexnvm properties.

Functions

- `status_t FLEXNVM_EepromWrite (flexnvm_config_t *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)`
Programs the EEPROM with data at locations passed in through parameters.

Initialization

- `status_t FLEXNVM_Init (flexnvm_config_t *config)`
Initializes the global flash properties structure members.

Erasing

- `status_t FLEXNVM_DflashErase (flexnvm_config_t *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)`
Erases the Dflash sectors encompassed by parameters passed into function.
- `status_t FLEXNVM_EraseAll (flexnvm_config_t *config, uint32_t key)`
Erases entire flexnvm.

Programming

- **status_t FLEXNVM_DflashProgram (flexnvm_config_t *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)**
Programs flash with data at locations passed in through parameters.
- **status_t FLEXNVM_ProgramPartition (flexnvm_config_t *config, ftfx_partition_flexram_load_opt_t option, uint32_t eepromDataSizeCode, uint32_t flexnvmPartitionCode)**
Prepares the FlexNVM block for use as data flash, EEPROM backup, or a combination of both and initializes the FlexRAM.

Reading

- **status_t FLEXNVM_ReadResource (flexnvm_config_t *config, uint32_t start, uint8_t *dst, uint32_t lengthInBytes, ftfx_read_resource_opt_t option)**
Reads the resource with data at locations passed in through parameters.

Verification

- **status_t FLEXNVM_DflashVerifyErase (flexnvm_config_t *config, uint32_t start, uint32_t lengthInBytes, ftfx_margin_value_t margin)**
Verifies an erasure of the desired flash area at a specified margin level.
- **status_t FLEXNVM_VerifyEraseAll (flexnvm_config_t *config, ftfx_margin_value_t margin)**
Verifies erasure of the entire flash at a specified margin level.
- **status_t FLEXNVM_DflashVerifyProgram (flexnvm_config_t *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, ftfx_margin_value_t margin, uint32_t *failedAddress, uint32_t *failedData)**
Verifies programming of the desired flash area at a specified margin level.

Security

- **status_t FLEXNVM_GetSecurityState (flexnvm_config_t *config, ftfx_security_state_t *state)**
Returns the security state via the pointer passed into the function.
- **status_t FLEXNVM_SecurityBypass (flexnvm_config_t *config, const uint8_t *backdoorKey)**
Allows users to bypass security with a backdoor key.

Flash Protection Utilities

- **status_t FLEXNVM_DflashSetProtection (flexnvm_config_t *config, uint8_t protectStatus)**
Sets the DFlash protection to the intended protection status.
- **status_t FLEXNVM_DflashGetProtection (flexnvm_config_t *config, uint8_t *protectStatus)**
Gets the DFlash protection status.
- **status_t FLEXNVM_EepromSetProtection (flexnvm_config_t *config, uint8_t protectStatus)**
Sets the EEPROM protection to the intended protection status.
- **status_t FLEXNVM_EepromGetProtection (flexnvm_config_t *config, uint8_t *protectStatus)**

Gets the EEPROM protection status.

Properties

- `status_t FLEXNVMGetProperty (flexnvm_config_t *config, flexnvm_property_tag_t whichProperty, uint32_t *value)`
Returns the desired flexnvm property.

14.4.2 Data Structure Documentation

14.4.2.1 struct flexnvm_config_t

An instance of this structure is allocated by the user of the Flexnvm driver and passed into each of the driver APIs.

14.4.3 Enumeration Type Documentation

14.4.3.1 enum flexnvm_property_tag_t

Enumerator

- `kFLEXNVM_PropertyDflashSectorSize` Dflash sector size property.
- `kFLEXNVM_PropertyDflashTotalSize` Dflash total size property.
- `kFLEXNVM_PropertyDflashBlockSize` Dflash block size property.
- `kFLEXNVM_PropertyDflashBlockCount` Dflash block count property.
- `kFLEXNVM_PropertyDflashBlockBaseAddr` Dflash block base address property.
- `kFLEXNVM_PropertyAliasDflashBlockBaseAddr` Dflash block base address Alias property.
- `kFLEXNVM_PropertyFlexRamBlockBaseAddr` FlexRam block base address property.
- `kFLEXNVM_PropertyFlexRamTotalSize` FlexRam total size property.
- `kFLEXNVM_PropertyEepromTotalSize` EEPROM total size property.

14.4.4 Function Documentation

14.4.4.1 status_t FLEXNVM_Init (`flexnvm_config_t * config`)

This function checks and initializes the Flash module for the other Flash APIs.

Parameters

<i>config</i>	Pointer to the storage for the driver runtime state.
---------------	--

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_PartitionStatusUpdateFailure</i>	Failed to update the partition status.

14.4.4.2 status_t FLEXNVM_DflashErase (*flexnvm_config_t * config, uint32_t start, uint32_t lengthInBytes, uint32_t key*)

This function erases the appropriate number of flash sectors based on the desired start address and length.

Parameters

<i>config</i>	The pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words) to be erased. Must be word-aligned.
<i>key</i>	The value used to validate all flash erase APIs.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the appropriate number of date flash sectors based on the desired start address and length were erased successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.

<i>kStatus_FTFx_AlignmentError</i>	The parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_AddressError</i>	The address is out of range.
<i>kStatus_FTFx_EraseKeyError</i>	The API erase key is invalid.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

14.4.4.3 status_t FLEXNVM_EraseAll (*flexnvm_config_t * config, uint32_t key*)

Parameters

<i>config</i>	Pointer to the storage for the driver runtime state.
<i>key</i>	A value used to validate all flash erase APIs.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the entire flexnvm has been erased successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_EraseKeyError</i>	API erase key is invalid.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.

<i>kStatus_FTFx_Access_Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during command execution.
<i>kStatus_FTFx_Partition_StatusUpdateFailure</i>	Failed to update the partition status.

14.4.4.4 **status_t FLEXNVM_DflashProgram (flexnvm_config_t * config, uint32_t start, uint8_t * src, uint32_t lengthInBytes)**

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be programmed. Must be word-aligned.
<i>src</i>	A pointer to the source buffer of data that is to be programmed into the flash.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the desired date have been successfully programed into specified date flash region.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.

<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_-CommandFailure</i>	Run-time error during the command execution.

14.4.4.5 status_t FLEXNVM_ProgramPartition (*flexnvm_config_t * config, ftfx_partition_flexram_load_opt_t option, uint32_t eepromDataSizeCode, uint32_t flexnvmPartitionCode*)

Parameters

<i>config</i>	Pointer to storage for the driver runtime state.
<i>option</i>	The option used to set FlexRAM load behavior during reset.
<i>eepromData-SizeCode</i>	Determines the amount of FlexRAM used in each of the available EEPROM subsystems.
<i>flexnvm-PartitionCode</i>	Specifies how to split the FlexNVM block between data flash memory and EEPROM backup memory supporting EEPROM functions.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the FlexNVM block for use as data flash, EEPROM backup, or a combination of both have been Prepared.
<i>kStatus_FTFx_Invalid-Argument</i>	Invalid argument is provided.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.

<i>kStatus_FTFx_CommandFailure</i>	Run-time error during command execution.
------------------------------------	--

14.4.4.6 status_t FLEXNVM_ReadResource (*flexnvm_config_t * config*, *uint32_t start*, *uint8_t * dst*, *uint32_t lengthInBytes*, *ftfx_read_resource_opt_t option*)

This function reads the flash memory with the desired location for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be programmed. Must be word-aligned.
<i>dst</i>	A pointer to the destination buffer of data that is used to store data to be read.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be read. Must be word-aligned.
<i>option</i>	The resource option which indicates which area should be read back.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the data have been read successfully from program flash IFR, data flash IFR space, and the Version ID field
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_-AlignmentError</i>	Parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.

<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.
------------------------------------	--

14.4.4.7 status_t FLEXNVM_DflashVerifyErase (flexnvm_config_t * config, uint32_t start, uint32_t lengthInBytes, ftfx_margin_value_t margin)

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
<i>margin</i>	Read margin choice.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the specified data flash region is in erased state.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.

<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.
------------------------------------	--

14.4.4.8 status_t FLEXNVM_VerifyEraseAll (*flexnvm_config_t * config, ftfx_margin_value_t margin*)

This function checks whether the flash is erased to the specified read margin level.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>margin</i>	Read margin choice.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the entire flexnvm region is in erased state.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

14.4.4.9 status_t FLEXNVM_DflashVerifyProgram (*flexnvm_config_t * config, uint32_t start, uint32_t lengthInBytes, const uint8_t * expectedData, ftfx_margin_value_t margin, uint32_t * failedAddress, uint32_t * failedData*)

This function verifies the data programmed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be verified. Must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
<i>expectedData</i>	A pointer to the expected data that is to be verified against.
<i>margin</i>	Read margin choice.
<i>failedAddress</i>	A pointer to the returned failing address.
<i>failedData</i>	A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the desired data have been programmed successfully into specified data flash region.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

14.4.4.10 status_t FLEXNVM_GetSecurityState (**flexnvm_config_t * config,** **ftfx_security_state_t * state**)

This function retrieves the current flash security status, including the security enabling state and the backdoor key enabling state.

Parameters

<i>config</i>	A pointer to storage for the driver runtime state.
<i>state</i>	A pointer to the value returned for the current security status code:

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the security state of flexnvm was stored to state.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.

14.4.4.11 **status_t FLEXNVM_SecurityBypass (flexnvm_config_t * *config*, const uint8_t * *backdoorKey*)**

If the MCU is in secured state, this function unsecures the MCU by comparing the provided backdoor key with ones in the flash configuration field.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>backdoorKey</i>	A pointer to the user buffer containing the backdoor key.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.

<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.
------------------------------------	--

14.4.4.12 status_t FLEXNVM_EepromWrite (*flexnvm_config_t * config, uint32_t start, uint8_t * src, uint32_t lengthInBytes*)

This function programs the emulated EEPROM with the desired data for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be programmed. Must be word-aligned.
<i>src</i>	A pointer to the source buffer of data that is to be programmed into the flash.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the desired data have been successfully programmed into specified eeprom region.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.
<i>kStatus_FTFx_SetFlexramAsEepromError</i>	Failed to set flexram as eeprom.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_RecoverFlexramAsRamError</i>	Failed to recover the FlexRAM as RAM.

14.4.4.13 status_t FLEXNVM_DflashSetProtection (*flexnvm_config_t * config, uint8_t protectStatus*)

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>protectStatus</i>	The expected protect status to set to the DFlash protection register. Each bit corresponds to the protection of the 1/8 of the total DFlash. The least significant bit corresponds to the lowest address area of the DFlash. The most significant bit corresponds to the highest address area of the DFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the specified DFlash region is protected.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_CommandNotSupported</i>	Flash API is not supported.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during command execution.

14.4.4.14 status_t FLEXNVM_DflashGetProtection (*flexnvm_config_t * config, uint8_t * protectStatus*)

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>protectStatus</i>	DFlash Protect status returned by the PFlash IP. Each bit corresponds to the protection of the 1/8 of the total DFlash. The least significant bit corresponds to the lowest address area of the DFlash. The most significant bit corresponds to the highest address area of the DFlash, and so on. There are two possible cases as below: 0: this area is protected. 1: this area is unprotected.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.

<i>kStatus_FTFx_-CommandNotSupported</i>	Flash API is not supported.
--	-----------------------------

14.4.4.15 status_t FLEXNVM_EepromSetProtection (**flexnvm_config_t * config, uint8_t protectStatus**)

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>protectStatus</i>	The expected protect status to set to the EEPROM protection register. Each bit corresponds to the protection of the 1/8 of the total EEPROM. The least significant bit corresponds to the lowest address area of the EEPROM. The most significant bit corresponds to the highest address area of EEPROM, and so on. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_-CommandNotSupported</i>	Flash API is not supported.
<i>kStatus_FTFx_-CommandFailure</i>	Run-time error during command execution.

14.4.4.16 status_t FLEXNVM_EepromGetProtection (**flexnvm_config_t * config, uint8_t * protectStatus**)

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>protectStatus</i>	DFlash Protect status returned by the PFlash IP. Each bit corresponds to the protection of the 1/8 of the total EEPROM. The least significant bit corresponds to the lowest address area of the EEPROM. The most significant bit corresponds to the highest address area of the EEPROM. There are two possible cases as below: 0: this area is protected. 1: this area is unprotected.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_CommandNotSupported</i>	Flash API is not supported.

14.4.4.17 status_t FLEXNVMGetProperty (*flexnvm_config_t * config,* *flexnvm_property_tag_t whichProperty, uint32_t * value*)

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>whichProperty</i>	The desired property from the list of properties in enum flexnvm_property_tag_t
<i>value</i>	A pointer to the value returned for the desired flexnvm property.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_UnknownProperty</i>	An unknown property tag.

14.5 fffx feature

14.5.1 Overview

Modules

- fffx adapter

Macros

- #define **FTFx_DRIVER_HAS_FLASH1_SUPPORT** (0U)
Indicates whether the secondary flash is supported in the Flash driver.

FTFx configuration

- #define **FTFx_DRIVER_IS_FLASH_RESIDENT** 1U
Flash driver location.
- #define **FTFx_DRIVER_IS_EXPORTED** 0U
Flash Driver Export option.

Secondary flash configuration

- #define **FTFx_FLASH1_HAS_PROT_CONTROL** (0U)
Indicates whether the secondary flash has its own protection register in flash module.
- #define **FTFx_FLASH1_HAS_XACC_CONTROL** (0U)
Indicates whether the secondary flash has its own Execute-Only access register in flash module.

14.5.2 Macro Definition Documentation

14.5.2.1 #define FTFx_DRIVER_IS_FLASH_RESIDENT 1U

Used for the flash resident application.

14.5.2.2 #define FTFx_DRIVER_IS_EXPORTED 0U

Used for the MCUXpresso SDK application.

14.5.2.3 #define FTFx_FLASH1_HAS_PROT_CONTROL (0U)

14.5.2.4 #define FTFx_FLASH1_HAS_XACC_CONTROL (0U)

14.5.3 ftx adapter

14.6 ftx controller

14.6.1 Overview

Modules

- [ftx utilities](#)

Data Structures

- struct [ftx_spec_mem_t](#)
ftx special memory access information. [More...](#)
- struct [ftx_mem_desc_t](#)
Flash memory descriptor. [More...](#)
- struct [ftx_ops_config_t](#)
Active FTFx information for the current operation. [More...](#)
- struct [ftx_ifr_desc_t](#)
Flash IFR memory descriptor. [More...](#)
- struct [ftx_config_t](#)
Flash driver state information. [More...](#)

Enumerations

- enum [ftx_partition_flexram_load_opt_t](#) {

kFTFx_PartitionFlexramLoadOptLoadedWithValidEepromData,

kFTFx_PartitionFlexramLoadOptNotLoaded = 0x01U }

Enumeration for the FlexRAM load during reset option.
- enum [ftx_read_resource_opt_t](#) {

kFTFx_ResourceOptionFlashIfr,

kFTFx_ResourceOptionVersionId = 0x01U }

Enumeration for the two possible options of flash read resource command.
- enum [ftx_margin_value_t](#) {

kFTFx_MarginValueNormal,

kFTFx_MarginValueUser,

kFTFx_MarginValueFactory,

kFTFx_MarginValueInvalid }

Enumeration for supported FTFx margin levels.
- enum [ftx_security_state_t](#) {

kFTFx_SecurityStateNotSecure = (int)0xc33cc33cu,

kFTFx_SecurityStateBackdoorEnabled = (int)0x5aa55aa5u,

kFTFx_SecurityStateBackdoorDisabled = (int)0x5ac33ca5u }

Enumeration for the three possible FTFx security states.
- enum [ftx_flexram_func_opt_t](#) {

kFTFx_FlexramFuncOptAvailableAsRam = 0xFFU,

kFTFx_FlexramFuncOptAvailableForEeprom = 0x00U }

Enumeration for the two possilbe options of set FlexRAM function command.

- enum `_flash_acceleration_ram_property`
Enumeration for acceleration ram property.
- enum `ftfx_swap_state_t` {

`kFTFx_SwapStateUninitialized` = 0x00U,
`kFTFx_SwapStateReady` = 0x01U,
`kFTFx_SwapStateUpdate` = 0x02U,
`kFTFx_SwapStateUpdateErased` = 0x03U,
`kFTFx_SwapStateComplete` = 0x04U,
`kFTFx_SwapStateDisabled` = 0x05U }
- Enumeration for the possible flash Swap status.*
- enum `_ftfx_memory_type`
Enumeration for FTFx memory type.

FTFx status

- enum {

`kStatus_FTFx_Success` = MAKE_STATUS(kStatusGroupGeneric, 0),
`kStatus_FTFx_InvalidArgument` = MAKE_STATUS(kStatusGroupGeneric, 4),
`kStatus_FTFx_SizeError` = MAKE_STATUS(kStatusGroupFtxDriver, 0),
`kStatus_FTFx_AlignmentError`,
`kStatus_FTFx_AddressError` = MAKE_STATUS(kStatusGroupFtxDriver, 2),
`kStatus_FTFx_AccessError`,
`kStatus_FTFx_ProtectionViolation`,
`kStatus_FTFx_CommandFailure`,
`kStatus_FTFx_UnknownProperty` = MAKE_STATUS(kStatusGroupFtxDriver, 6),
`kStatus_FTFx_EraseKeyError` = MAKE_STATUS(kStatusGroupFtxDriver, 7),
`kStatus_FTFx_RegionExecuteOnly` = MAKE_STATUS(kStatusGroupFtxDriver, 8),
`kStatus_FTFx_ExecuteInRamFunctionNotReady`,
`kStatus_FTFx_PartitionStatusUpdateFailure`,
`kStatus_FTFx_SetFlexramAsEepromError`,
`kStatus_FTFx_RecoverFlexramAsRamError`,
`kStatus_FTFx_SetFlexramAsRamError` = MAKE_STATUS(kStatusGroupFtxDriver, 13),
`kStatus_FTFx_RecoverFlexramAsEepromError`,
`kStatus_FTFx_CommandNotSupported` = MAKE_STATUS(kStatusGroupFtxDriver, 15),
`kStatus_FTFx_SwapSystemNotInUninitialized`,
`kStatus_FTFx_SwapIndicatorAddressError`,
`kStatus_FTFx_ReadOnlyProperty` = MAKE_STATUS(kStatusGroupFtxDriver, 18),
`kStatus_FTFx_InvalidPropertyValue`,
`kStatus_FTFx_InvalidSpeculationOption`,
`kStatus_FTFx_CommandOperationInProgress` }
- FTFx driver status codes.*
- #define `kStatusGroupGeneric` 0
FTFx driver status group.
- #define `kStatusGroupFtxDriver` 1

FTFx API key

- enum `_ftfx_driver_api_keys` { `kFTFx_ApiEraseKey` = FOUR_CHAR_CODE('k', 'f', 'e', 'k') }
- Enumeration for FTFx driver API keys.*

Initialization

- void `FTFx_API_Init` (`ftfx_config_t` *config)
- Initializes the global flash properties structure members.*

Erasing

- `status_t FTFx_CMD_Erase` (`ftfx_config_t` *config, `uint32_t` start, `uint32_t` lengthInBytes, `uint32_t` key)

Erases the flash sectors encompassed by parameters passed into function.

- `status_t FTFx_CMD_EraseSectorNonBlocking` (`ftfx_config_t` *config, `uint32_t` start, `uint32_t` key)

Erases the flash sectors encompassed by parameters passed into function.

- `status_t FTFx_CMD_EraseAll` (`ftfx_config_t` *config, `uint32_t` key)

Erases entire flash.

- `status_t FTFx_CMD_EraseAllExecuteOnlySegments` (`ftfx_config_t` *config, `uint32_t` key)

Erases all program flash execute-only segments defined by the FXACC registers.

Programming

- `status_t FTFx_CMD_Program` (`ftfx_config_t` *config, `uint32_t` start, `const uint8_t` *src, `uint32_t` lengthInBytes)

Programs flash with data at locations passed in through parameters.

- `status_t FTFx_CMD_ProgramOnce` (`ftfx_config_t` *config, `uint32_t` index, `const uint8_t` *src, `uint32_t` lengthInBytes)

Programs Program Once Field through parameters.

Reading

- `status_t FTFx_CMD_ReadOnce` (`ftfx_config_t` *config, `uint32_t` index, `uint8_t` *dst, `uint32_t` lengthInBytes)

Reads the Program Once Field through parameters.

- `status_t FTFx_CMD_ReadResource` (`ftfx_config_t` *config, `uint32_t` start, `uint8_t` *dst, `uint32_t` lengthInBytes, `ftfx_read_resource_opt_t` option)

Reads the resource with data at locations passed in through parameters.

Verification

- `status_t FTFx_CMD_VerifyErase (ftfx_config_t *config, uint32_t start, uint32_t lengthInBytes, ftx_margin_value_t margin)`
Verifies an erasure of the desired flash area at a specified margin level.
- `status_t FTFx_CMD_VerifyEraseAll (ftfx_config_t *config, ftx_margin_value_t margin)`
Verifies erasure of the entire flash at a specified margin level.
- `status_t FTFx_CMD_VerifyEraseAllExecuteOnlySegments (ftfx_config_t *config, ftx_margin_value_t margin)`
Verifies whether the program flash execute-only segments have been erased to the specified read margin level.
- `status_t FTFx_CMD_VerifyProgram (ftfx_config_t *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, ftx_margin_value_t margin, uint32_t *failedAddress, uint32_t *failedData)`
Verifies programming of the desired flash area at a specified margin level.

Security

- `status_t FTFx_REG_GetSecurityState (ftfx_config_t *config, ftx_security_state_t *state)`
Returns the security state via the pointer passed into the function.
- `status_t FTFx_CMD_SecurityBypass (ftfx_config_t *config, const uint8_t *backdoorKey)`
Allows users to bypass security with a backdoor key.

14.6.2 Data Structure Documentation

14.6.2.1 struct ftx_spec_mem_t

Data Fields

- `uint32_t base`
Base address of flash special memory.
- `uint32_t size`
size of flash special memory.
- `uint32_t count`
flash special memory count.

Field Documentation

- (1) `uint32_t ftx_spec_mem_t::base`
- (2) `uint32_t ftx_spec_mem_t::size`
- (3) `uint32_t ftx_spec_mem_t::count`

14.6.2.2 struct ftfx_mem_desc_t

Data Fields

- `uint32_t blockBase`
A base address of the flash block.
- `uint32_t totalSize`
The size of the flash block.
- `uint32_t sectorSize`
The size in bytes of a sector of flash.
- `uint32_t blockCount`
A number of flash blocks.
- `uint8_t type`
Type of flash block.
- `uint8_t index`
Index of flash block.

Field Documentation

- (1) `uint8_t ftfx_mem_desc_t::type`
- (2) `uint8_t ftfx_mem_desc_t::index`
- (3) `uint32_t ftfx_mem_desc_t::totalSize`
- (4) `uint32_t ftfx_mem_desc_t::sectorSize`
- (5) `uint32_t ftfx_mem_desc_t::blockCount`

14.6.2.3 struct ftfx_ops_config_t

Data Fields

- `uint32_t convertedAddress`
A converted address for the current flash type.

Field Documentation

- (1) `uint32_t ftfx_ops_config_t::convertedAddress`

14.6.2.4 struct ftfx_ifr_desc_t

14.6.2.5 struct ftfx_config_t

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

Data Fields

- `uint32_t flexramBlockBase`
The base address of the FlexRAM/acceleration RAM.
- `uint32_t flexramTotalSize`
The size of the FlexRAM/acceleration RAM.
- `uint16_t eepromTotalSize`
The size of EEPROM area which was partitioned from FlexRAM.
- `function_ptr_t runCmdFuncAddr`
An buffer point to the flash execute-in-RAM function.

Field Documentation

(1) `function_ptr_t ftfx_config_t::runCmdFuncAddr`

14.6.3 Macro Definition Documentation

14.6.3.1 #define kStatusGroupGeneric 0

14.6.4 Enumeration Type Documentation

14.6.4.1 anonymous enum

Enumerator

`kStatus_FTFx_Success` API is executed successfully.

`kStatus_FTFx_InvalidArgument` Invalid argument.

`kStatus_FTFx_SizeError` Error size.

`kStatus_FTFx_AlignmentError` Parameter is not aligned with the specified baseline.

`kStatus_FTFx_AddressError` Address is out of range.

`kStatus_FTFx_AccessError` Invalid instruction codes and out-of bound addresses.

`kStatus_FTFx_ProtectionViolation` The program/erase operation is requested to execute on protected areas.

`kStatus_FTFx_CommandFailure` Run-time error during command execution.

`kStatus_FTFx_UnknownProperty` Unknown property.

`kStatus_FTFx_EraseKeyError` API erase key is invalid.

`kStatus_FTFx_RegionExecuteOnly` The current region is execute-only.

`kStatus_FTFx_ExecuteInRamFunctionNotReady` Execute-in-RAM function is not available.

`kStatus_FTFx_PartitionStatusUpdateFailure` Failed to update partition status.

`kStatus_FTFx_SetFlexramAsEepromError` Failed to set FlexRAM as EEPROM.

`kStatus_FTFx_RecoverFlexramAsRamError` Failed to recover FlexRAM as RAM.

`kStatus_FTFx_SetFlexramAsRamError` Failed to set FlexRAM as RAM.

`kStatus_FTFx_RecoverFlexramAsEepromError` Failed to recover FlexRAM as EEPROM.

`kStatus_FTFx_CommandNotSupported` Flash API is not supported.

`kStatus_FTFx_SwapSystemNotInUninitialized` Swap system is not in an uninitialized state.

`kStatus_FTFx_SwapIndicatorAddressError` The swap indicator address is invalid.

`kStatus_FTFx_ReadOnlyProperty` The flash property is read-only.

kStatus_FTFx_InvalidPropertyValue The flash property value is out of range.

kStatus_FTFx_InvalidSpeculationOption The option of flash prefetch speculation is invalid.

kStatus_FTFx_CommandOperationInProgress The option of flash command is processing.

14.6.4.2 enum _ftfx_driver_api_keys

Note

The resulting value is built with a byte order such that the string being readable in expected order when viewed in a hex editor, if the value is treated as a 32-bit little endian value.

Enumerator

kFTFx_ApiEraseKey Key value used to validate all FTFx erase APIs.

14.6.4.3 enum ftfx_partition_flexram_load_opt_t

Enumerator

kFTFx_PartitionFlexramLoadOptLoadedWithValidEepromData FlexRAM is loaded with valid EEPROM data during reset sequence.

kFTFx_PartitionFlexramLoadOptNotLoaded FlexRAM is not loaded during reset sequence.

14.6.4.4 enum ftfx_read_resource_opt_t

Enumerator

kFTFx_ResourceOptionFlashIfr Select code for Program flash 0 IFR, Program flash swap 0 IFR, Data flash 0 IFR.

kFTFx_ResourceOptionVersionId Select code for the version ID.

14.6.4.5 enum ftfx_margin_value_t

Enumerator

kFTFx_MarginValueNormal Use the 'normal' read level for 1s.

kFTFx_MarginValueUser Apply the 'User' margin to the normal read-1 level.

kFTFx_MarginValueFactory Apply the 'Factory' margin to the normal read-1 level.

kFTFx_MarginValueInvalid Not real margin level, Used to determine the range of valid margin level.

14.6.4.6 enum ftfx_security_state_t

Enumerator

kFTFx_SecurityStateNotSecure Flash is not secure.

kFTFx_SecurityStateBackdoorEnabled Flash backdoor is enabled.

kFTFx_SecurityStateBackdoorDisabled Flash backdoor is disabled.

14.6.4.7 enum ftfx_flexram_func_opt_t

Enumerator

kFTFx_FlexramFuncOptAvailableAsRam An option used to make FlexRAM available as RAM.

kFTFx_FlexramFuncOptAvailableForEeprom An option used to make FlexRAM available for E-EPROM.

14.6.4.8 enum ftfx_swap_state_t

Enumerator

kFTFx_SwapStateUninitialized Flash Swap system is in an uninitialized state.

kFTFx_SwapStateReady Flash Swap system is in a ready state.

kFTFx_SwapStateUpdate Flash Swap system is in an update state.

kFTFx_SwapStateUpdateErased Flash Swap system is in an updateErased state.

kFTFx_SwapStateComplete Flash Swap system is in a complete state.

kFTFx_SwapStateDisabled Flash Swap system is in a disabled state.

14.6.5 Function Documentation

14.6.5.1 void FTFx_API_Init (ftfx_config_t * config)

This function checks and initializes the Flash module for the other Flash APIs.

Parameters

<i>config</i>	Pointer to the storage for the driver runtime state.
---------------	--

14.6.5.2 status_t FTFx_CMD_Erase (ftfx_config_t * config, uint32_t start, uint32_t lengthInBytes, uint32_t key)

This function erases the appropriate number of flash sectors based on the desired start address and length.

Parameters

<i>config</i>	The pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words) to be erased. Must be word-aligned.
<i>key</i>	The value used to validate all flash erase APIs.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	The parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_AddressError</i>	The address is out of range.
<i>kStatus_FTFx_EraseKeyError</i>	The API erase key is invalid.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

14.6.5.3 status_t FTFx_CMD_EraseSectorNonBlocking (*ftfx_config_t * config, uint32_t start, uint32_t key*)

This function erases one flash sector size based on the start address.

Parameters

<i>config</i>	The pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
<i>key</i>	The value used to validate all flash erase APIs.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	The parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_AddressError</i>	The address is out of range.
<i>kStatus_FTFx_EraseKeyError</i>	The API erase key is invalid.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.

14.6.5.4 status_t FTFx_CMD_EraseAll (*ftfx_config_t * config, uint32_t key*)

Parameters

<i>config</i>	Pointer to the storage for the driver runtime state.
<i>key</i>	A value used to validate all flash erase APIs.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_EraseKeyError</i>	API erase key is invalid.

<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_-CommandFailure</i>	Run-time error during command execution.
<i>kStatus_FTFx_Partition-StatusUpdateFailure</i>	Failed to update the partition status.

14.6.5.5 **status_t FTFx_CMD_EraseAllExecuteOnlySegments (ftfx_config_t * config, uint32_t key)**

Parameters

<i>config</i>	Pointer to the storage for the driver runtime state.
<i>key</i>	A value used to validate all flash erase APIs.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_EraseKey-Error</i>	API erase key is invalid.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.

<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.
------------------------------------	--

14.6.5.6 **status_t FTFx_CMD_Program (ftfx_config_t * config, uint32_t start, const uint8_t * src, uint32_t lengthInBytes)**

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be programmed. Must be word-aligned.
<i>src</i>	A pointer to the source buffer of data that is to be programmed into the flash.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.

<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.
------------------------------------	--

14.6.5.7 status_t FTFx_CMD_ProgramOnce (ftfx_config_t * config, uint32_t index, const uint8_t * src, uint32_t lengthInBytes)

This function programs the Program Once Field with the desired data for a given flash area as determined by the index and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>index</i>	The index indicating which area of the Program Once Field to be programmed.
<i>src</i>	A pointer to the source buffer of data that is to be programmed into the Program Once Field.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

14.6.5.8 status_t FTFx_CMD_ReadOnce (ftfx_config_t * config, uint32_t index, uint8_t * dst, uint32_t lengthInBytes)

This function reads the read once feild with given index and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>index</i>	The index indicating the area of program once field to be read.
<i>dst</i>	A pointer to the destination buffer of data that is used to store data to be read.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_-CommandFailure</i>	Run-time error during the command execution.

14.6.5.9 status_t FTFx_CMD_ReadResource (*ftfx_config_t * config*, *uint32_t start*, *uint8_t * dst*, *uint32_t lengthInBytes*, *ftfx_read_resource_opt_t option*)

This function reads the flash memory with the desired location for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be programmed. Must be word-aligned.
<i>dst</i>	A pointer to the destination buffer of data that is used to store data to be read.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be read. Must be word-aligned.

<i>option</i>	The resource option which indicates which area should be read back.
---------------	---

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

14.6.5.10 `status_t FTFx_CMD_VerifyErase(ftfx_config_t * config, uint32_t start, uint32_t lengthInBytes, ftfx_margin_value_t margin)`

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
<i>margin</i>	Read margin choice.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

14.6.5.11 `status_t FTFx_CMD_VerifyEraseAll(ftfx_config_t * config, ftfx_margin_value_t margin)`

This function checks whether the flash is erased to the specified read margin level.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>margin</i>	Read margin choice.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.

<i>kStatus_FTFx_Access_Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

14.6.5.12 status_t FTFx_CMD_VerifyEraseAllExecuteOnlySegments (*ftfx_config_t * config, ftfx_margin_value_t margin*)

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>margin</i>	Read margin choice.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access_Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

14.6.5.13 status_t FTFx_CMD_VerifyProgram (*ftfx_config_t * config, uint32_t start, uint32_t lengthInBytes, const uint8_t * expectedData, ftfx_margin_value_t margin, uint32_t * failedAddress, uint32_t * failedData*)

This function verifies the data programed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be verified. Must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
<i>expectedData</i>	A pointer to the expected data that is to be verified against.
<i>margin</i>	Read margin choice.
<i>failedAddress</i>	A pointer to the returned failing address.
<i>failedData</i>	A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

14.6.5.14 **status_t FTFx_REG_GetSecurityState (ftfx_config_t * config, ftfx_security_state_t * state)**

This function retrieves the current flash security status, including the security enabling state and the backdoor key enabling state.

Parameters

<i>config</i>	A pointer to storage for the driver runtime state.
<i>state</i>	A pointer to the value returned for the current security status code:

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.

14.6.5.15 **status_t FTFx_CMD_SecurityBypass (*ftfx_config_t * config, const uint8_t * backdoorKey*)**

If the MCU is in secured state, this function unsecures the MCU by comparing the provided backdoor key with ones in the flash configuration field.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>backdoorKey</i>	A pointer to the user buffer containing the backdoor key.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_-CommandFailure</i>	Run-time error during the command execution.

14.6.6 ftx utilities

14.6.6.1 Overview

Macros

- `#define MAKE_VERSION(major, minor, bugfix) (((major) << 16) | ((minor) << 8) | (bugfix))`
Constructs the version number for drivers.
- `#define MAKE_STATUS(group, code) (((group)*100) + (code)))`
Constructs a status code value from a group and a code number.
- `#define FOUR_CHAR_CODE(a, b, c, d) (((uint32_t)(d) << 24u) | ((uint32_t)(c) << 16u) | ((uint32_t)(b) << 8u) | ((uint32_t)(a)))`
Constructs the four character code for the Flash driver API key.
- `#define B1P4(b) (((uint32_t)(b)&0xFFU) << 24U)`
bytes2word utility.

Alignment macros

- `#define ALIGN_DOWN(x, a) (((uint32_t)(x)) & ~((uint32_t)(a)-1u))`
Alignment(down) utility.
- `#define ALIGN_UP(x, a) ALIGN_DOWN((uint32_t)(x) + (uint32_t)(a)-1u, a)`
Alignment(up) utility.

14.6.6.2 Macro Definition Documentation

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

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

14.6.6.2.3 `#define FOUR_CHAR_CODE(a, b, c, d) (((uint32_t)(d) << 24u) | ((uint32_t)(c) << 16u) | ((uint32_t)(b) << 8u) | ((uint32_t)(a)))`

14.6.6.2.4 `#define ALIGN_DOWN(x, a) (((uint32_t)(x)) & ~((uint32_t)(a)-1u))`

14.6.6.2.5 `#define ALIGN_UP(x, a) ALIGN_DOWN((uint32_t)(x) + (uint32_t)(a)-1u, a)`

14.6.6.2.6 `#define B1P4(b) (((uint32_t)(b)&0xFFU) << 24U)`

Chapter 15

FlexBus: External Bus Interface Driver

15.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Crossbar External Bus Interface (FlexBus) block of MCUXpresso SDK devices.

A multifunction external bus interface is provided on the device with a basic functionality to interface to slave-only devices. It can be directly connected to the following asynchronous or synchronous devices with little or no additional circuitry.

- External ROMs
- Flash memories
- Programmable logic devices
- Other simple target (slave) devices

For asynchronous devices, a simple chip-select based interface can be used. The FlexBus interface has up to six general purpose chip-selects, FB_CS[5:0]. The number of chip selects available depends on the device and its pin configuration.

15.2 FlexBus functional operation

To configure the FlexBus driver, use one of the two ways to configure the `flexbus_config_t` structure.

1. Using the `FLEXBUS_GetDefaultConfig()` function.
2. Set parameters in the `flexbus_config_t` structure.

To initialize and configure the FlexBus driver, call the `FLEXBUS_Init()` function and pass a pointer to the `flexbus_config_t` structure.

To de-initialize the FlexBus driver, call the `FLEXBUS_Deinit()` function.

15.3 Typical use case and example

This example shows how to write/read to external memory (MRAM) by using the FlexBus module.

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

Data Structures

- struct `flexbus_config_t`
Configuration structure that the user needs to set. [More...](#)

Enumerations

- enum `flexbus_port_size_t` {
 `kFLEXBUS_4Bytes` = 0x00U,
 `kFLEXBUS_1Byte` = 0x01U,

- ```
kFLEXBUS_2Bytes = 0x02U }
```

*Defines port size for FlexBus peripheral.*
- enum `flexbus_write_address_hold_t` {
 

```
kFLEXBUS_Hold1Cycle = 0x00U,
```

```
kFLEXBUS_Hold2Cycles = 0x01U,
```

```
kFLEXBUS_Hold3Cycles = 0x02U,
```

```
kFLEXBUS_Hold4Cycles = 0x03U }
```

*Defines number of cycles to hold address and attributes for FlexBus peripheral.*
- enum `flexbus_read_address_hold_t` {
 

```
kFLEXBUS_Hold1Or0Cycles = 0x00U,
```

```
kFLEXBUS_Hold2Or1Cycles = 0x01U,
```

```
kFLEXBUS_Hold3Or2Cycle = 0x02U,
```

```
kFLEXBUS_Hold4Or3Cycle = 0x03U }
```

*Defines number of cycles to hold address and attributes for FlexBus peripheral.*
- enum `flexbus_address_setup_t` {
 

```
kFLEXBUS_FirstRisingEdge = 0x00U,
```

```
kFLEXBUS_SecondRisingEdge = 0x01U,
```

```
kFLEXBUS_ThirdRisingEdge = 0x02U,
```

```
kFLEXBUS_FourthRisingEdge = 0x03U }
```

*Address setup for FlexBus peripheral.*
- enum `flexbus_bytelane_shift_t` {
 

```
kFLEXBUS_NotShifted = 0x00U,
```

```
kFLEXBUS_Shifted = 0x01U }
```

*Defines byte-lane shift for FlexBus peripheral.*
- enum `flexbus_multiplex_group1_t` {
 

```
kFLEXBUS_MultiplexGroup1_FB_ALE = 0x00U,
```

```
kFLEXBUS_MultiplexGroup1_FB_CS1 = 0x01U,
```

```
kFLEXBUS_MultiplexGroup1_FB_TS = 0x02U }
```

*Defines multiplex group1 valid signals.*
- enum `flexbus_multiplex_group2_t` {
 

```
kFLEXBUS_MultiplexGroup2_FB_CS4 = 0x00U,
```

```
kFLEXBUS_MultiplexGroup2_FB_TSIZ0 = 0x01U,
```

```
kFLEXBUS_MultiplexGroup2_FB_BE_31_24 = 0x02U }
```

*Defines multiplex group2 valid signals.*
- enum `flexbus_multiplex_group3_t` {
 

```
kFLEXBUS_MultiplexGroup3_FB_CS5 = 0x00U,
```

```
kFLEXBUS_MultiplexGroup3_FB_TSIZ1 = 0x01U,
```

```
kFLEXBUS_MultiplexGroup3_FB_BE_23_16 = 0x02U }
```

*Defines multiplex group3 valid signals.*
- enum `flexbus_multiplex_group4_t` {
 

```
kFLEXBUS_MultiplexGroup4_FB_TBST = 0x00U,
```

```
kFLEXBUS_MultiplexGroup4_FB_CS2 = 0x01U,
```

```
kFLEXBUS_MultiplexGroup4_FB_BE_15_8 = 0x02U }
```

*Defines multiplex group4 valid signals.*
- enum `flexbus_multiplex_group5_t` {
 

```
kFLEXBUS_MultiplexGroup5_FB_TA = 0x00U,
```

```
kFLEXBUS_MultiplexGroup5_FB_CS3 = 0x01U,
```

```
kFLEXBUS_MultiplexGroup5_FB_BE_7_0 = 0x02U }
```

*Defines multiplex group5 valid signals.*

## Driver version

- #define **FSL\_FLEXBUS\_DRIVER\_VERSION** (MAKE\_VERSION(2, 1, 1))  
*Version 2.1.1.*

## FlexBus functional operation

- void **FLEXBUS\_Init** (FB\_Type \*base, const flexbus\_config\_t \*config)  
*Initializes and configures the FlexBus module.*
- void **FLEXBUS\_Deinit** (FB\_Type \*base)  
*De-initializes a FlexBus instance.*
- void **FLEXBUS\_GetDefaultConfig** (flexbus\_config\_t \*config)  
*Initializes the FlexBus configuration structure.*

## 15.4 Data Structure Documentation

### 15.4.1 struct flexbus\_config\_t

#### Data Fields

- uint8\_t **chip**  
*Chip FlexBus for validation.*
- uint8\_t **waitStates**  
*Value of wait states.*
- uint8\_t **secondaryWaitStates**  
*Value of secondary wait states.*
- uint32\_t **chipBaseAddress**  
*Chip base address for using FlexBus.*
- uint32\_t **chipBaseAddressMask**  
*Chip base address mask.*
- bool **writeProtect**  
*Write protected.*
- bool **burstWrite**  
*Burst-Write enable.*
- bool **burstRead**  
*Burst-Read enable.*
- bool **byteEnableMode**  
*Byte-enable mode support.*
- bool **autoAcknowledge**  
*Auto acknowledge setting.*
- bool **extendTransferAddress**  
*Extend transfer start/extend address latch enable.*
- bool **secondaryWaitStatesEnable**  
*Enable secondary wait states.*
- **flexbus\_port\_size\_t portSize**  
*Port size of transfer.*
- **flexbus\_bytelane\_shift\_t byteLaneShift**

- **`flexbus_write_address_hold_t`** `writeAddressHold`  
*Byte-lane shift enable.*  
    Write address hold or deselect option.
- **`flexbus_read_address_hold_t`** `readAddressHold`  
*Read address hold or deselect option.*
- **`flexbus_address_setup_t`** `addressSetup`  
*Address setup setting.*
- **`flexbus_multiplex_group1_t`** `group1MultiplexControl`  
*FlexBus Signal Group 1 Multiplex control.*
- **`flexbus_multiplex_group2_t`** `group2MultiplexControl`  
*FlexBus Signal Group 2 Multiplex control.*
- **`flexbus_multiplex_group3_t`** `group3MultiplexControl`  
*FlexBus Signal Group 3 Multiplex control.*
- **`flexbus_multiplex_group4_t`** `group4MultiplexControl`  
*FlexBus Signal Group 4 Multiplex control.*
- **`flexbus_multiplex_group5_t`** `group5MultiplexControl`  
*FlexBus Signal Group 5 Multiplex control.*

## 15.5 Macro Definition Documentation

### 15.5.1 `#define FSL_FLEXBUS_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`

## 15.6 Enumeration Type Documentation

### 15.6.1 `enum flexbus_port_size_t`

Enumerator

**`kFLEXBUS_4Bytes`** 32-bit port size  
**`kFLEXBUS_1Byte`** 8-bit port size  
**`kFLEXBUS_2Bytes`** 16-bit port size

### 15.6.2 `enum flexbus_write_address_hold_t`

Enumerator

**`kFLEXBUS_Hold1Cycle`** Hold address and attributes one cycles after FB\_CSn negates on writes.  
**`kFLEXBUS_Hold2Cycles`** Hold address and attributes two cycles after FB\_CSn negates on writes.  
**`kFLEXBUS_Hold3Cycles`** Hold address and attributes three cycles after FB\_CSn negates on writes.  
**`kFLEXBUS_Hold4Cycles`** Hold address and attributes four cycles after FB\_CSn negates on writes.

### 15.6.3 `enum flexbus_read_address_hold_t`

Enumerator

**`kFLEXBUS_Hold1Or0Cycles`** Hold address and attributes 1 or 0 cycles on reads.

***kFLEXBUS\_Hold2Or1Cycles*** Hold address and attributes 2 or 1 cycles on reads.

***kFLEXBUS\_Hold3Or2Cycle*** Hold address and attributes 3 or 2 cycles on reads.

***kFLEXBUS\_Hold4Or3Cycle*** Hold address and attributes 4 or 3 cycles on reads.

#### 15.6.4 enum flexbus\_address\_setup\_t

Enumerator

***kFLEXBUS\_FirstRisingEdge*** Assert FB\_CSn on first rising clock edge after address is asserted.

***kFLEXBUS\_SecondRisingEdge*** Assert FB\_CSn on second rising clock edge after address is asserted.

***kFLEXBUS\_ThirdRisingEdge*** Assert FB\_CSn on third rising clock edge after address is asserted.

***kFLEXBUS\_FourthRisingEdge*** Assert FB\_CSn on fourth rising clock edge after address is asserted.

#### 15.6.5 enum flexbus\_bytelane\_shift\_t

Enumerator

***kFLEXBUS\_NotShifted*** Not shifted. Data is left-justified on FB\_AD

***kFLEXBUS\_Shifted*** Shifted. Data is right justified on FB\_AD

#### 15.6.6 enum flexbus\_multiplex\_group1\_t

Enumerator

***kFLEXBUS\_MultiplexGroup1\_FB\_ALE*** FB\_ALE.

***kFLEXBUS\_MultiplexGroup1\_FB\_CS1*** FB\_CS1.

***kFLEXBUS\_MultiplexGroup1\_FB\_TS*** FB\_TS.

#### 15.6.7 enum flexbus\_multiplex\_group2\_t

Enumerator

***kFLEXBUS\_MultiplexGroup2\_FB\_CS4*** FB\_CS4.

***kFLEXBUS\_MultiplexGroup2\_FB\_TSIZ0*** FB\_TSIZ0.

***kFLEXBUS\_MultiplexGroup2\_FB\_BE\_31\_24*** FB\_BE\_31\_24.

### 15.6.8 enum flexbus\_multiplex\_group3\_t

Enumerator

*kFLEXBUS\_MultiplexGroup3\_FB\_CS5* FB\_CS5.  
*kFLEXBUS\_MultiplexGroup3\_FB\_TSIZ1* FB\_TSIZ1.  
*kFLEXBUS\_MultiplexGroup3\_FB\_BE\_23\_16* FB\_BE\_23\_16.

### 15.6.9 enum flexbus\_multiplex\_group4\_t

Enumerator

*kFLEXBUS\_MultiplexGroup4\_FB\_TBST* FB\_TBST.  
*kFLEXBUS\_MultiplexGroup4\_FB\_CS2* FB\_CS2.  
*kFLEXBUS\_MultiplexGroup4\_FB\_BE\_15\_8* FB\_BE\_15\_8.

### 15.6.10 enum flexbus\_multiplex\_group5\_t

Enumerator

*kFLEXBUS\_MultiplexGroup5\_FB\_TA* FB\_TA.  
*kFLEXBUS\_MultiplexGroup5\_FB\_CS3* FB\_CS3.  
*kFLEXBUS\_MultiplexGroup5\_FB\_BE\_7\_0* FB\_BE\_7\_0.

## 15.7 Function Documentation

### 15.7.1 void FLEXBUS\_Init ( **FB\_Type** \* *base*, **const flexbus\_config\_t** \* *config* )

This function enables the clock gate for FlexBus module. Only chip 0 is validated and set to known values. Other chips are disabled. Note that in this function, certain parameters, depending on external memories, must be set before using the [FLEXBUS\\_Init\(\)](#) function. This example shows how to set up the `uart_state_t` and the `flexbus_config_t` parameters and how to call the `FLEXBUS_Init` function by passing in these parameters.

```
flexbus_config_t flexbusConfig;
FLEXBUS_GetDefaultConfig(&flexbusConfig);
flexbusConfig.waitStates = 2U;
flexbusConfig.chipBaseAddress = 0x60000000U;
flexbusConfig.chipBaseAddressMask = 7U;
FLEXBUS_Init(FB, &flexbusConfig);
```

## Parameters

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

**15.7.2 void FLEXBUS\_Deinit ( FB\_Type \* *base* )**

This function disables the clock gate of the FlexBus module clock.

## Parameters

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

**15.7.3 void FLEXBUS\_GetDefaultConfig ( flexbus\_config\_t \* *config* )**

This function initializes the FlexBus configuration structure to default value. The default values are.

```

fbConfig->chip = 0;
fbConfig->writeProtect = 0;
fbConfig->burstWrite = 0;
fbConfig->burstRead = 0;
fbConfig->byteEnableMode = 0;
fbConfig->autoAcknowledge = true;
fbConfig->extendTransferAddress = 0;
fbConfig->secondaryWaitStates = 0;
fbConfig->byteLaneShift = kFLEXBUS_NotShifted;
fbConfig->writeAddressHold = kFLEXBUS_Hold1Cycle;
fbConfig->readAddressHold = kFLEXBUS_Hold1Or0Cycles;
fbConfig->addressSetup = kFLEXBUS_FirstRisingEdge;
fbConfig->portSize = kFLEXBUS_1Byte;
fbConfig->group1MultiplexControl = kFLEXBUS_MultiplexGroup1_FB_ALE;
fbConfig->group2MultiplexControl = kFLEXBUS_MultiplexGroup2_FB_CS4 ;
fbConfig->group3MultiplexControl = kFLEXBUS_MultiplexGroup3_FB_CS5;
fbConfig->group4MultiplexControl = kFLEXBUS_MultiplexGroup4_FB_TBST;
fbConfig->group5MultiplexControl = kFLEXBUS_MultiplexGroup5_FB_TA;

```

## Parameters

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

## See Also

[FLEXBUS\\_Init](#)

# Chapter 16

## FTM: FlexTimer Driver

### 16.1 Overview

The MCUXpresso SDK provides a driver for the FlexTimer Module (FTM) of MCUXpresso SDK devices.

### 16.2 Function groups

The FTM driver supports the generation of PWM signals, input capture, dual edge capture, output compare, and quadrature decoder modes. The driver also supports configuring each of the FTM fault inputs.

#### 16.2.1 Initialization and deinitialization

The function `FTM_Init()` initializes the FTM with specified configurations. The function `FTM_GetDefaultConfig()` gets the default configurations. The initialization function configures the FTM for the requested register update mode for registers with buffers. It also sets up the FTM's fault operation mode and FTM behavior in the BDM mode.

The function `FTM_Deinit()` disables the FTM counter and turns off the module clock.

#### 16.2.2 PWM Operations

The function `FTM_SetupPwm()` sets up FTM channels for the PWM output. The function sets up the PWM signal properties for multiple channels. Each channel has its own duty cycle and level-mode specified. However, the same PWM period and PWM mode 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 `FTM_UpdatePwmDutycycle()` updates the PWM signal duty cycle of a particular FTM channel.

The function `FTM_UpdateChnlEdgeLevelSelect()` updates the level select bits of a particular FTM channel. This can be used to disable the PWM output when making changes to the PWM signal.

#### 16.2.3 Input capture operations

The function `FTM_SetupInputCapture()` sets up an FTM channel for the input capture. The user can specify the capture edge and a filter value to be used when processing the input signal.

The function [FTM\\_SetupDualEdgeCapture\(\)](#) can be used to measure the pulse width of a signal. A channel pair is used during capture with the input signal coming through a channel n. The user can specify whether to use one-shot or continuous capture, the capture edge for each channel, and any filter value to be used when processing the input signal.

#### 16.2.4 Output compare operations

The function [FTM\\_SetupOutputCompare\(\)](#) sets up an FTM channel for the output comparison. The user can specify the channel output on a successful comparison and a comparison value.

#### 16.2.5 Quad decode

The function [FTM\\_SetupQuadDecode\(\)](#) sets up FTM channels 0 and 1 for quad decoding. The user can specify the quad decoding mode, polarity, and filter properties for each input signal.

#### 16.2.6 Fault operation

The function [FTM\\_SetupFault\(\)](#) sets up the properties for each fault. The user can specify the fault polarity and whether to use a filter on a fault input. The overall fault filter value and fault control mode are set up during initialization.

### 16.3 Register Update

Some of the FTM registers have buffers. The driver supports various methods to update these registers with the content of the register buffer. The registers can be updated using the PWM synchronized loading or an intermediate point loading. The update mechanism for register with buffers can be specified through the following fields available in the configuration structure. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/fmMultiple PWM synchronization update modes can be used by providing an OR'ed list of options available in the enumeration [ftm\\_pwm\\_sync\\_method\\_t](#) to the pwmSyncMode field.

When using an intermediate reload points, the PWM synchronization is not required. Multiple reload points can be used by providing an OR'ed list of options available in the enumeration [ftm\\_reload\\_point\\_t](#) to the reloadPoints field.

The driver initialization function sets up the appropriate bits in the FTM module based on the register update options selected.

If software PWM synchronization is used, the below function can be used to initiate a software trigger. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/fm

### 16.4 Typical use case

### 16.4.1 PWM output

Output a PWM signal on two FTM channels with different duty cycles. Periodically update the PWM signal duty cycle. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/ftm

## Data Structures

- struct `ftm_chnl_pwm_signal_param_t`  
*Options to configure a FTM channel's PWM signal.* [More...](#)
- struct `ftm_chnl_pwm_config_param_t`  
*Options to configure a FTM channel using precise setting.* [More...](#)
- struct `ftm_dual_edge_capture_param_t`  
*FlexTimer dual edge capture parameters.* [More...](#)
- struct `ftm_phase_params_t`  
*FlexTimer quadrature decode phase parameters.* [More...](#)
- struct `ftm_fault_param_t`  
*Structure is used to hold the parameters to configure a FTM fault.* [More...](#)
- struct `ftm_config_t`  
*FTM configuration structure.* [More...](#)

## Enumerations

- enum `ftm_chnl_t` {
   
kFTM\_Chnl\_0 = 0U,
   
kFTM\_Chnl\_1,
   
kFTM\_Chnl\_2,
   
kFTM\_Chnl\_3,
   
kFTM\_Chnl\_4,
   
kFTM\_Chnl\_5,
   
kFTM\_Chnl\_6,
   
kFTM\_Chnl\_7 }
   
*List of FTM channels.*
- enum `ftm_fault_input_t` {
   
kFTM\_Fault\_0 = 0U,
   
kFTM\_Fault\_1,
   
kFTM\_Fault\_2,
   
kFTM\_Fault\_3 }
   
*List of FTM faults.*
- enum `ftm_pwm_mode_t` {
   
kFTM\_EdgeAlignedPwm = 0U,
   
kFTM\_CenterAlignedPwm,
   
kFTM\_EdgeAlignedCombinedPwm,
   
kFTM\_CenterAlignedCombinedPwm,
   
kFTM\_AsymmetricalCombinedPwm }
   
*FTM PWM operation modes.*
- enum `ftm_pwm_level_select_t` {

```
kFTM_NoPwmSignal = 0U,
kFTM_LowTrue,
kFTM_HighTrue }
```

*FTM PWM output pulse mode: high-true, low-true or no output.*

- enum `ftm_output_compare_mode_t` {
   
kFTM\_NoOutputSignal = (1U << FTM\_CnSC\_MSA\_SHIFT),
   
kFTM\_ToggleOnMatch = ((1U << FTM\_CnSC\_MSA\_SHIFT) | (1U << FTM\_CnSC\_ELSA\_S-HIFT)),
   
kFTM\_ClearOnMatch = ((1U << FTM\_CnSC\_MSA\_SHIFT) | (2U << FTM\_CnSC\_ELSA\_SHIFT)),
   
kFTM\_SetOnMatch = ((1U << FTM\_CnSC\_MSA\_SHIFT) | (3U << FTM\_CnSC\_ELSA\_SHIFT)) }

*FlexTimer output compare mode.*

- enum `ftm_input_capture_edge_t` {
   
kFTM\_RisingEdge = (1U << FTM\_CnSC\_ELSA\_SHIFT),
   
kFTM\_FallingEdge = (2U << FTM\_CnSC\_ELSA\_SHIFT),
   
kFTM\_RiseAndFallEdge = (3U << FTM\_CnSC\_ELSA\_SHIFT) }

*FlexTimer input capture edge.*

- enum `ftm_dual_edge_capture_mode_t` {
   
kFTM\_OneShot = 0U,
   
kFTM\_Continuous = (1U << FTM\_CnSC\_MSA\_SHIFT) }

*FlexTimer dual edge capture modes.*

- enum `ftm_quad_decode_mode_t` {
   
kFTM\_QuadPhaseEncode = 0U,
   
kFTM\_QuadCountAndDir }
  
- enum `ftm_phase_polarity_t` {
   
kFTM\_QuadPhaseNormal = 0U,
   
kFTM\_QuadPhaseInvert }

*FlexTimer quadrature phase polarities.*

- enum `ftm_deadtime_prescale_t` {
   
kFTM\_Deadtime\_Prescale\_1 = 1U,
   
kFTM\_Deadtime\_Prescale\_4,
   
kFTM\_Deadtime\_Prescale\_16 }

*FlexTimer pre-scaler factor for the dead time insertion.*

- enum `ftm_clock_source_t` {
   
kFTM\_SystemClock = 1U,
   
kFTM\_FixedClock,
   
kFTM\_ExternalClock }

*FlexTimer clock source selection.*

- enum `ftm_clock_prescale_t` {

```
kFTM_Prescale_Divide_1 = 0U,
kFTM_Prescale_Divide_2,
kFTM_Prescale_Divide_4,
kFTM_Prescale_Divide_8,
kFTM_Prescale_Divide_16,
kFTM_Prescale_Divide_32,
kFTM_Prescale_Divide_64,
kFTM_Prescale_Divide_128 }
```

*FlexTimer pre-scaler factor selection for the clock source.*

- enum `ftm_bdm_mode_t` {
   
kFTM\_BdmMode\_0 = 0U,
   
kFTM\_BdmMode\_1,
   
kFTM\_BdmMode\_2,
   
kFTM\_BdmMode\_3 }

*Options for the FlexTimer behaviour in BDM Mode.*

- enum `ftm_fault_mode_t` {
   
kFTM\_Fault\_Disable = 0U,
   
kFTM\_Fault\_EvenChnls,
   
kFTM\_Fault\_AllChnlsMan,
   
kFTM\_Fault\_AllChnlsAuto }

*Options for the FTM fault control mode.*

- enum `ftm_external_trigger_t` {
   
kFTM\_Chnl0Trigger = (1U << 4),
   
kFTM\_Chnl1Trigger = (1U << 5),
   
kFTM\_Chnl2Trigger = (1U << 0),
   
kFTM\_Chnl3Trigger = (1U << 1),
   
kFTM\_Chnl4Trigger = (1U << 2),
   
kFTM\_Chnl5Trigger = (1U << 3),
   
kFTM\_InitTrigger = (1U << 6) }

*FTM external trigger options.*

- enum `ftm_pwm_sync_method_t` {
   
kFTM\_SoftwareTrigger = FTM\_SYNC\_SWSYNC\_MASK,
   
kFTM\_HardwareTrigger\_0 = FTM\_SYNC\_TRIG0\_MASK,
   
kFTM\_HardwareTrigger\_1 = FTM\_SYNC\_TRIG1\_MASK,
   
kFTM\_HardwareTrigger\_2 = FTM\_SYNC\_TRIG2\_MASK }

*FlexTimer PWM sync options to update registers with buffer.*

- enum `ftm_reload_point_t` {

```

kFTM_Chnl0Match = (1U << 0),
kFTM_Chnl1Match = (1U << 1),
kFTM_Chnl2Match = (1U << 2),
kFTM_Chnl3Match = (1U << 3),
kFTM_Chnl4Match = (1U << 4),
kFTM_Chnl5Match = (1U << 5),
kFTM_Chnl6Match = (1U << 6),
kFTM_Chnl7Match = (1U << 7),
kFTM_CntMax = (1U << 8),
kFTM_CntMin = (1U << 9),
kFTM_HalfCycMatch = (1U << 10) }

```

*FTM options available as loading point for register reload.*

- enum `ftm_interrupt_enable_t` {

```

kFTM_Chnl0InterruptEnable = (1U << 0),
kFTM_Chnl1InterruptEnable = (1U << 1),
kFTM_Chnl2InterruptEnable = (1U << 2),
kFTM_Chnl3InterruptEnable = (1U << 3),
kFTM_Chnl4InterruptEnable = (1U << 4),
kFTM_Chnl5InterruptEnable = (1U << 5),
kFTM_Chnl6InterruptEnable = (1U << 6),
kFTM_Chnl7InterruptEnable = (1U << 7),
kFTM_FaultInterruptEnable = (1U << 8),
kFTM_TimeOverflowInterruptEnable = (1U << 9),
kFTM_ReloadInterruptEnable = (1U << 10) }

```

*List of FTM interrupts.*

- enum `ftm_status_flags_t` {

```

kFTM_Chnl0Flag = (1U << 0),
kFTM_Chnl1Flag = (1U << 1),
kFTM_Chnl2Flag = (1U << 2),
kFTM_Chnl3Flag = (1U << 3),
kFTM_Chnl4Flag = (1U << 4),
kFTM_Chnl5Flag = (1U << 5),
kFTM_Chnl6Flag = (1U << 6),
kFTM_Chnl7Flag = (1U << 7),
kFTM_FaultFlag = (1U << 8),
kFTM_TimeOverflowFlag = (1U << 9),
kFTM_ChnlTriggerFlag = (1U << 10),
kFTM_ReloadFlag = (1U << 11) }

```

*List of FTM flags.*

- enum {

```

kFTM_QuadDecoderCountingIncreaseFlag = FTM_QDCTRL_QUADIR_MASK,
kFTM_QuadDecoderCountingOverflowOnTopFlag = FTM_QDCTRL_TOFDIR_MASK }

```

*List of FTM Quad Decoder flags.*

## Functions

- void **FTM\_SetupFaultInput** (FTM\_Type \*base, **ftm\_fault\_input\_t** faultNumber, const **ftm\_fault\_param\_t** \*faultParams)
 

*Sets up the working of the FTM fault inputs protection.*
- static void **FTM\_SetGlobalTimeBaseOutputEnable** (FTM\_Type \*base, bool enable)
 

*Enables or disables the FTM global time base signal generation to other FTMs.*
- static void **FTM\_SetOutputMask** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber, bool mask)
 

*Sets the FTM peripheral timer channel output mask.*
- static void **FTM\_SetSoftwareTrigger** (FTM\_Type \*base, bool enable)
 

*Enables or disables the FTM software trigger for PWM synchronization.*
- static void **FTM\_SetWriteProtection** (FTM\_Type \*base, bool enable)
 

*Enables or disables the FTM write protection.*
- static void **FTM\_EnableDmaTransfer** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber, bool enable)
 

*Enable DMA transfer or not.*

## Driver version

- #define **FSL\_FTM\_DRIVER\_VERSION** (MAKE\_VERSION(2, 5, 0))
 

*FTM driver version 2.5.0.*

## Initialization and deinitialization

- **status\_t FTM\_Init** (FTM\_Type \*base, const **ftm\_config\_t** \*config)
 

*Ungates the FTM clock and configures the peripheral for basic operation.*
- void **FTM\_Deinit** (FTM\_Type \*base)
 

*Gates the FTM clock.*
- void **FTM\_GetDefaultConfig** (**ftm\_config\_t** \*config)
 

*Fills in the FTM configuration structure with the default settings.*
- static **ftm\_clock\_prescale\_t FTM\_CalculateCounterClkDiv** (FTM\_Type \*base, uint32\_t counterPeriod\_Hz, uint32\_t srcClock\_Hz)
 

*brief Calculates the counter clock prescaler.*

## Channel mode operations

- **status\_t FTM\_SetupPwm** (FTM\_Type \*base, const **ftm\_chnl\_pwm\_signal\_param\_t** \*chnlParams, uint8\_t numOfChnls, **ftm\_pwm\_mode\_t** mode, uint32\_t pwmFreq\_Hz, uint32\_t srcClock\_Hz)
 

*Configures the PWM signal parameters.*
- **status\_t FTM\_UpdatePwmDutyCycle** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber, **ftm\_pwm\_mode\_t** currentPwmMode, uint8\_t dutyCyclePercent)
 

*Updates the duty cycle of an active PWM signal.*
- void **FTM\_UpdateChnlEdgeLevelSelect** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber, uint8\_t level)
 

*Updates the edge level selection for a channel.*
- **status\_t FTM\_SetupPwmMode** (FTM\_Type \*base, const **ftm\_chnl\_pwm\_config\_param\_t** \*chnlParams, uint8\_t numOfChnls, **ftm\_pwm\_mode\_t** mode)
 

*Configures the PWM mode parameters.*
- void **FTM\_SetupInputCapture** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber, **ftm\_input\_capture\_edge\_t** captureMode, uint32\_t filterValue)
 

*Enables capturing an input signal on the channel using the function parameters.*
- void **FTM\_SetupOutputCompare** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber, **ftm\_output\_compare\_mode\_t** compareMode, uint32\_t compareValue)

- Configures the FTM to generate timed pulses.
- void **FTM\_SetupDualEdgeCapture** (FTM\_Type \*base, **ftm\_chnl\_t** chnlPairNumber, const **ftm\_dual\_edge\_capture\_param\_t** \*edgeParam, uint32\_t filterValue)  
*Configures the dual edge capture mode of the FTM.*

## Interrupt Interface

- void **FTM\_EnableInterrupts** (FTM\_Type \*base, uint32\_t mask)  
*Enables the selected FTM interrupts.*
- void **FTM\_DisableInterrupts** (FTM\_Type \*base, uint32\_t mask)  
*Disables the selected FTM interrupts.*
- uint32\_t **FTM\_GetEnabledInterrupts** (FTM\_Type \*base)  
*Gets the enabled FTM interrupts.*

## Status Interface

- uint32\_t **FTM\_GetStatusFlags** (FTM\_Type \*base)  
*Gets the FTM status flags.*
- void **FTM\_ClearStatusFlags** (FTM\_Type \*base, uint32\_t mask)  
*Clears the FTM status flags.*

## Read and write the timer period

- static void **FTM\_SetTimerPeriod** (FTM\_Type \*base, uint32\_t ticks)  
*Sets the timer period in units of ticks.*
- static uint32\_t **FTM\_GetCurrentTimerCount** (FTM\_Type \*base)  
*Reads the current timer counting value.*
- static uint32\_t **FTM\_GetInputCaptureValue** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber)  
*Reads the captured value.*

## Timer Start and Stop

- static void **FTM\_StartTimer** (FTM\_Type \*base, **ftm\_clock\_source\_t** clockSource)  
*Starts the FTM counter.*
- static void **FTM\_StopTimer** (FTM\_Type \*base)  
*Stops the FTM counter.*

## Software output control

- static void **FTM\_SetSoftwareCtrlEnable** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber, bool value)  
*Enables or disables the channel software output control.*
- static void **FTM\_SetSoftwareCtrlVal** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber, bool value)  
*Sets the channel software output control value.*

## Channel pair operations

- static void **FTM\_SetFaultControlEnable** (FTM\_Type \*base, **ftm\_chnl\_t** chnlPairNumber, bool value)  
*This function enables/disables the fault control in a channel pair.*
- static void **FTM\_SetDeadTimeEnable** (FTM\_Type \*base, **ftm\_chnl\_t** chnlPairNumber, bool value)

- This function enables/disables the dead time insertion in a channel pair.
- static void **FTM\_SetComplementaryEnable** (FTM\_Type \*base, **ftm\_chnl\_t** chnlPairNumber, bool value)
 

*This function enables/disables complementary mode in a channel pair.*
- static void **FTM\_SetInvertEnable** (FTM\_Type \*base, **ftm\_chnl\_t** chnlPairNumber, bool value)
 

*This function enables/disables inverting control in a channel pair.*

## Quad Decoder

- void **FTM\_SetupQuadDecode** (FTM\_Type \*base, const **ftm\_phase\_params\_t** \*phaseAParams, const **ftm\_phase\_params\_t** \*phaseBParams, **ftm\_quad\_decode\_mode\_t** quadMode)
 

*Configures the parameters and activates the quadrature decoder mode.*
- static uint32\_t **FTM\_GetQuadDecoderFlags** (FTM\_Type \*base)
 

*Gets the FTM Quad Decoder flags.*
- static void **FTM\_SetQuadDecoderModuloValue** (FTM\_Type \*base, uint32\_t startValue, uint32\_t overValue)
 

*Sets the modulo values for Quad Decoder.*
- static uint32\_t **FTM\_GetQuadDecoderCounterValue** (FTM\_Type \*base)
 

*Gets the current Quad Decoder counter value.*
- static void **FTM\_ClearQuadDecoderCounterValue** (FTM\_Type \*base)
 

*Clears the current Quad Decoder counter value.*

## 16.5 Data Structure Documentation

### 16.5.1 struct **ftm\_chnl\_pwm\_signal\_param\_t**

#### Data Fields

- **ftm\_chnl\_t chnlNumber**

*The channel/channel pair number.*
- **ftm\_pwm\_level\_select\_t level**

*PWM output active level select.*
- **uint8\_t dutyCyclePercent**

*PWM pulse width, value should be between 0 to 100 0 = inactive signal(0% duty cycle)...*
- **uint8\_t firstEdgeDelayPercent**

*Used only in kFTM\_AsymmetricalCombinedPwm mode to generate an asymmetrical PWM.*
- **bool enableComplementary**

*Used only in combined PWM mode.*
- **bool enableDeadtime**

*Used only in combined PWM mode with enable complementary.*

#### Field Documentation

##### (1) **ftm\_chnl\_t ftm\_chnl\_pwm\_signal\_param\_t::chnlNumber**

In combined mode, this represents the channel pair number.

##### (2) **ftm\_pwm\_level\_select\_t ftm\_chnl\_pwm\_signal\_param\_t::level**

**(3) uint8\_t ftm\_chnl\_pwm\_signal\_param\_t::dutyCyclePercent**

100 = always active signal (100% duty cycle).

**(4) uint8\_t ftm\_chnl\_pwm\_signal\_param\_t::firstEdgeDelayPercent**

Specifies the delay to the first edge in a PWM period. If unsure leave as 0; Should be specified as a percentage of the PWM period

**(5) bool ftm\_chnl\_pwm\_signal\_param\_t::enableComplementary**

true: The combined channels output complementary signals; false: The combined channels output same signals;

**(6) bool ftm\_chnl\_pwm\_signal\_param\_t::enableDeadtime**

true: The deadtime insertion in this pair of channels is enabled; false: The deadtime insertion in this pair of channels is disabled.

**16.5.2 struct ftm\_chnl\_pwm\_config\_param\_t****Data Fields**

- **ftm\_chnl\_t chnlNumber**  
*The channel/channel pair number.*
- **ftm\_pwm\_level\_select\_t level**  
*PWM output active level select.*
- **uint16\_t dutyValue**  
*PWM pulse width, the uint of this value is timer ticks.*
- **uint16\_t firstEdgeValue**  
*Used only in kFTM\_AsymmetricalCombinedPwm mode to generate an asymmetrical PWM.*
- **bool enableComplementary**  
*Used only in combined PWM mode.*
- **bool enableDeadtime**  
*Used only in combined PWM mode with enable complementary.*

**Field Documentation****(1) ftm\_chnl\_t ftm\_chnl\_pwm\_config\_param\_t::chnlNumber**

In combined mode, this represents the channel pair number.

**(2) ftm\_pwm\_level\_select\_t ftm\_chnl\_pwm\_config\_param\_t::level****(3) uint16\_t ftm\_chnl\_pwm\_config\_param\_t::dutyValue****(4) uint16\_t ftm\_chnl\_pwm\_config\_param\_t::firstEdgeValue**

Specifies the delay to the first edge in a PWM period. If unsure leave as 0, uint of this value is timer ticks.

**(5) bool ftm\_chnl\_pwm\_config\_param\_t::enableComplementary**

true: The combined channels output complementary signals; false: The combined channels output same signals;

**(6) bool ftm\_chnl\_pwm\_config\_param\_t::enableDeadtime**

true: The deadtime insertion in this pair of channels is enabled; false: The deadtime insertion in this pair of channels is disabled.

**16.5.3 struct ftm\_dual\_edge\_capture\_param\_t****Data Fields**

- `ftm_dual_edge_capture_mode_t mode`  
*Dual Edge Capture mode.*
- `ftm_input_capture_edge_t currChanEdgeMode`  
*Input capture edge select for channel n.*
- `ftm_input_capture_edge_t nextChanEdgeMode`  
*Input capture edge select for channel n+1.*

**16.5.4 struct ftm\_phase\_params\_t****Data Fields**

- `bool enablePhaseFilter`  
*True: enable phase filter; false: disable filter.*
- `uint32_t phaseFilterVal`  
*Filter value, used only if phase filter is enabled.*
- `ftm_phase_polarity_t phasePolarity`  
*Phase polarity.*

**16.5.5 struct ftm\_fault\_param\_t****Data Fields**

- `bool enableFaultInput`  
*True: Fault input is enabled; false: Fault input is disabled.*
- `bool faultLevel`  
*True: Fault polarity is active low; in other words, '0' indicates a fault; False: Fault polarity is active high.*
- `bool useFaultFilter`  
*True: Use the filtered fault signal; False: Use the direct path from fault input.*

### 16.5.6 struct `ftm_config_t`

This structure holds the configuration settings for the FTM peripheral. To initialize this structure to reasonable defaults, call the [FTM\\_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

- **`ftm_clock_prescale_t prescale`**  
*FTM clock prescale value.*
- **`ftm_bdm_mode_t bdmMode`**  
*FTM behavior in BDM mode.*
- **`uint32_t pwmSyncMode`**  
*Synchronization methods to use to update buffered registers; Multiple update modes can be used by providing an OR'ed list of options available in enumeration [ftm\\_pwm\\_sync\\_method\\_t](#).*
- **`uint32_t reloadPoints`**  
*FTM reload points; When using this, the PWM synchronization is not required.*
- **`ftm_fault_mode_t faultMode`**  
*FTM fault control mode.*
- **`uint8_t faultFilterValue`**  
*Fault input filter value.*
- **`ftm_deadtime_prescale_t deadTimePrescale`**  
*The dead time prescalar value.*
- **`uint32_t deadTimeValue`**  
*The dead time value deadTimeValue's available range is 0-1023 when register has DTVALEX, otherwise its available range is 0-63.*
- **`uint32_t extTriggers`**  
*External triggers to enable.*
- **`uint8_t chnlInitState`**  
*Defines the initialization value of the channels in OUTINT register.*
- **`uint8_t chnlPolarity`**  
*Defines the output polarity of the channels in POL register.*
- **`bool useGlobalTimeBase`**  
*True: Use of an external global time base is enabled; False: disabled.*

### Field Documentation

(1) **`uint32_t ftm_config_t::pwmSyncMode`**

(2) **`uint32_t ftm_config_t::reloadPoints`**

Multiple reload points can be used by providing an OR'ed list of options available in enumeration [ftm\\_reload\\_point\\_t](#).

(3) **`uint32_t ftm_config_t::deadTimeValue`**

**(4) uint32\_t ftm\_config\_t::extTriggers**

Multiple trigger sources can be enabled by providing an OR'ed list of options available in enumeration [ftm\\_external\\_trigger\\_t](#).

**16.6 Macro Definition Documentation****16.6.1 #define FSL\_FTM\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 0))****16.7 Enumeration Type Documentation****16.7.1 enum ftm\_chnl\_t**

Note

Actual number of available channels is SoC dependent

Enumerator

- kFTM\_Chnl\_0*** FTM channel number 0.
- kFTM\_Chnl\_1*** FTM channel number 1.
- kFTM\_Chnl\_2*** FTM channel number 2.
- kFTM\_Chnl\_3*** FTM channel number 3.
- kFTM\_Chnl\_4*** FTM channel number 4.
- kFTM\_Chnl\_5*** FTM channel number 5.
- kFTM\_Chnl\_6*** FTM channel number 6.
- kFTM\_Chnl\_7*** FTM channel number 7.

**16.7.2 enum ftm\_fault\_input\_t**

Enumerator

- kFTM\_Fault\_0*** FTM fault 0 input pin.
- kFTM\_Fault\_1*** FTM fault 1 input pin.
- kFTM\_Fault\_2*** FTM fault 2 input pin.
- kFTM\_Fault\_3*** FTM fault 3 input pin.

**16.7.3 enum ftm\_pwm\_mode\_t**

Enumerator

- kFTM\_EdgeAlignedPwm*** Edge-aligned PWM.
- kFTM\_CenterAlignedPwm*** Center-aligned PWM.
- kFTM\_EdgeAlignedCombinedPwm*** Edge-aligned combined PWM.

*kFTM\_CenterAlignedCombinedPwm* Center-aligned combined PWM.

*kFTM\_AsymmetricalCombinedPwm* Asymmetrical combined PWM.

#### 16.7.4 enum ftm\_pwm\_level\_select\_t

Enumerator

*kFTM\_NoPwmSignal* No PWM output on pin.

*kFTM\_LowTrue* Low true pulses.

*kFTM\_HighTrue* High true pulses.

#### 16.7.5 enum ftm\_output\_compare\_mode\_t

Enumerator

*kFTM\_NoOutputSignal* No channel output when counter reaches CnV.

*kFTM\_ToggleOnMatch* Toggle output.

*kFTM\_ClearOnMatch* Clear output.

*kFTM\_SetOnMatch* Set output.

#### 16.7.6 enum ftm\_input\_capture\_edge\_t

Enumerator

*kFTM\_RisingEdge* Capture on rising edge only.

*kFTM\_FallingEdge* Capture on falling edge only.

*kFTM\_RiseAndFallEdge* Capture on rising or falling edge.

#### 16.7.7 enum ftm\_dual\_edge\_capture\_mode\_t

Enumerator

*kFTM\_OneShot* One-shot capture mode.

*kFTM\_Continuous* Continuous capture mode.

#### 16.7.8 enum ftm\_quad\_decode\_mode\_t

Enumerator

*kFTM\_QuadPhaseEncode* Phase A and Phase B encoding mode.

*kFTM\_QuadCountAndDir* Count and direction encoding mode.

**16.7.9 enum ftm\_phase\_polarity\_t**

Enumerator

*kFTM\_QuadPhaseNormal* Phase input signal is not inverted.*kFTM\_QuadPhaseInvert* Phase input signal is inverted.**16.7.10 enum ftm\_deadtime\_prescale\_t**

Enumerator

*kFTM\_Deadtime\_Prescale\_1* Divide by 1.*kFTM\_Deadtime\_Prescale\_4* Divide by 4.*kFTM\_Deadtime\_Prescale\_16* Divide by 16.**16.7.11 enum ftm\_clock\_source\_t**

Enumerator

*kFTM\_SystemClock* System clock selected.*kFTM\_FixedClock* Fixed frequency clock.*kFTM\_ExternalClock* External clock.**16.7.12 enum ftm\_clock\_prescale\_t**

Enumerator

*kFTM\_Prescale\_Divide\_1* Divide by 1.*kFTM\_Prescale\_Divide\_2* Divide by 2.*kFTM\_Prescale\_Divide\_4* Divide by 4.*kFTM\_Prescale\_Divide\_8* Divide by 8.*kFTM\_Prescale\_Divide\_16* Divide by 16.*kFTM\_Prescale\_Divide\_32* Divide by 32.*kFTM\_Prescale\_Divide\_64* Divide by 64.*kFTM\_Prescale\_Divide\_128* Divide by 128.**16.7.13 enum ftm\_bdm\_mode\_t**

Enumerator

*kFTM\_BdmMode\_0* FTM counter stopped, CH(n)F bit can be set, FTM channels in functional mode, writes to MOD,CNTIN and C(n)V registers bypass the register buffers.

***kFTM\_BdmMode\_1*** FTM counter stopped, CH(n)F bit is not set, FTM channels outputs are forced to their safe value , writes to MOD,CNTIN and C(n)V registers bypass the register buffers.

***kFTM\_BdmMode\_2*** FTM counter stopped, CH(n)F bit is not set, FTM channels outputs are frozen when chip enters in BDM mode, writes to MOD,CNTIN and C(n)V registers bypass the register buffers.

***kFTM\_BdmMode\_3*** FTM counter in functional mode, CH(n)F bit can be set, FTM channels in functional mode, writes to MOD,CNTIN and C(n)V registers is in fully functional mode.

#### 16.7.14 enum ftm\_fault\_mode\_t

Enumerator

***kFTM\_Fault\_Disable*** Fault control is disabled for all channels.

***kFTM\_Fault\_EvenChnls*** Enabled for even channels only(0,2,4,6) with manual fault clearing.

***kFTM\_Fault\_AllChnlsMan*** Enabled for all channels with manual fault clearing.

***kFTM\_Fault\_AllChnlsAuto*** Enabled for all channels with automatic fault clearing.

#### 16.7.15 enum ftm\_external\_trigger\_t

Note

Actual available external trigger sources are SoC-specific

Enumerator

***kFTM\_Chnl0Trigger*** Generate trigger when counter equals chnl 0 CnV reg.

***kFTM\_Chnl1Trigger*** Generate trigger when counter equals chnl 1 CnV reg.

***kFTM\_Chnl2Trigger*** Generate trigger when counter equals chnl 2 CnV reg.

***kFTM\_Chnl3Trigger*** Generate trigger when counter equals chnl 3 CnV reg.

***kFTM\_Chnl4Trigger*** Generate trigger when counter equals chnl 4 CnV reg.

***kFTM\_Chnl5Trigger*** Generate trigger when counter equals chnl 5 CnV reg.

***kFTM\_InitTrigger*** Generate Trigger when counter is updated with CNTIN.

#### 16.7.16 enum ftm\_pwm\_sync\_method\_t

Enumerator

***kFTM\_SoftwareTrigger*** Software triggers PWM sync.

***kFTM\_HardwareTrigger\_0*** Hardware trigger 0 causes PWM sync.

***kFTM\_HardwareTrigger\_1*** Hardware trigger 1 causes PWM sync.

***kFTM\_HardwareTrigger\_2*** Hardware trigger 2 causes PWM sync.

**16.7.17 enum ftm\_reload\_point\_t**

Note

Actual available reload points are SoC-specific

Enumerator

- kFTM\_Chnl0Match*** Channel 0 match included as a reload point.
- kFTM\_Chnl1Match*** Channel 1 match included as a reload point.
- kFTM\_Chnl2Match*** Channel 2 match included as a reload point.
- kFTM\_Chnl3Match*** Channel 3 match included as a reload point.
- kFTM\_Chnl4Match*** Channel 4 match included as a reload point.
- kFTM\_Chnl5Match*** Channel 5 match included as a reload point.
- kFTM\_Chnl6Match*** Channel 6 match included as a reload point.
- kFTM\_Chnl7Match*** Channel 7 match included as a reload point.
- kFTM\_CntMax*** Use in up-down count mode only, reload when counter reaches the maximum value.
- kFTM\_CntMin*** Use in up-down count mode only, reload when counter reaches the minimum value.
- kFTM\_HalfCycMatch*** Available on certain SoC's, half cycle match reload point.

**16.7.18 enum ftm\_interrupt\_enable\_t**

Note

Actual available interrupts are SoC-specific

Enumerator

- kFTM\_Chnl0InterruptEnable*** Channel 0 interrupt.
- kFTM\_Chnl1InterruptEnable*** Channel 1 interrupt.
- kFTM\_Chnl2InterruptEnable*** Channel 2 interrupt.
- kFTM\_Chnl3InterruptEnable*** Channel 3 interrupt.
- kFTM\_Chnl4InterruptEnable*** Channel 4 interrupt.
- kFTM\_Chnl5InterruptEnable*** Channel 5 interrupt.
- kFTM\_Chnl6InterruptEnable*** Channel 6 interrupt.
- kFTM\_Chnl7InterruptEnable*** Channel 7 interrupt.
- kFTM\_FaultInterruptEnable*** Fault interrupt.
- kFTM\_TimeOverflowInterruptEnable*** Time overflow interrupt.
- kFTM\_ReloadInterruptEnable*** Reload interrupt; Available only on certain SoC's.

**16.7.19 enum ftm\_status\_flags\_t**

Note

Actual available flags are SoC-specific

Enumerator

|                              |                                               |
|------------------------------|-----------------------------------------------|
| <i>kFTM_Chnl0Flag</i>        | Channel 0 Flag.                               |
| <i>kFTM_Chnl1Flag</i>        | Channel 1 Flag.                               |
| <i>kFTM_Chnl2Flag</i>        | Channel 2 Flag.                               |
| <i>kFTM_Chnl3Flag</i>        | Channel 3 Flag.                               |
| <i>kFTM_Chnl4Flag</i>        | Channel 4 Flag.                               |
| <i>kFTM_Chnl5Flag</i>        | Channel 5 Flag.                               |
| <i>kFTM_Chnl6Flag</i>        | Channel 6 Flag.                               |
| <i>kFTM_Chnl7Flag</i>        | Channel 7 Flag.                               |
| <i>kFTM_FaultFlag</i>        | Fault Flag.                                   |
| <i>kFTM_TimeOverflowFlag</i> | Time overflow Flag.                           |
| <i>kFTM_ChnlTriggerFlag</i>  | Channel trigger Flag.                         |
| <i>kFTM_ReloadFlag</i>       | Reload Flag; Available only on certain SoC's. |

### 16.7.20 anonymous enum

Enumerator

|                                                  |                                                                                           |
|--------------------------------------------------|-------------------------------------------------------------------------------------------|
| <i>kFTM_QuadDecoderCountingIncreaseFlag</i>      | Counting direction is increasing (FTM counter increment), or the direction is decreasing. |
| <i>kFTM_QuadDecoderCountingOverflowOnTopFlag</i> | Indicates if the TOF bit was set on the top or the bottom of counting.                    |

## 16.8 Function Documentation

### 16.8.1 status\_t FTM\_Init ( *FTM\_Type* \* *base*, *const ftm\_config\_t* \* *config* )

Note

This API should be called at the beginning of the application which is using the FTM driver. If the FTM instance has only TPM features, please use the TPM driver.

Parameters

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

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

Returns

kStatus\_Success indicates success; Else indicates failure.

### 16.8.2 void FTM\_Deinit ( FTM\_Type \* *base* )

Parameters

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

### 16.8.3 void FTM\_GetDefaultConfig ( ftm\_config\_t \* *config* )

The default values are:

```
* config->prescale = kFTM_Prescale_Divide_1;
* config->bdmMode = kFTM_BdmMode_0;
* config->pwmSyncMode = kFTM_SoftwareTrigger;
* config->reloadPoints = 0;
* config->faultMode = kFTM_Fault_Disable;
* config->faultFilterValue = 0;
* config->deadTimePrescale = kFTM_Deadtime_Prescale_1;
* config->deadTimeValue = 0;
* config->extTriggers = 0;
* config->chnlInitState = 0;
* config->chnlPolarity = 0;
* config->useGlobalTimeBase = false;
*
```

Parameters

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

### 16.8.4 static ftm\_clock\_prescale\_t FTM\_CalculateCounterClkDiv ( FTM\_Type \* *base*, uint32\_t *counterPeriod\_Hz*, uint32\_t *srcClock\_Hz* ) [inline], [static]

This function calculates the values for SC[PS] bit.

param *base* FTM peripheral base address param *counterPeriod\_Hz* The desired frequency in Hz which corresponding to the time when the counter reaches the mod value param *srcClock\_Hz* FTM counter clock in Hz

return Calculated clock prescaler value, see [ftm\\_clock\\_prescale\\_t](#).

**16.8.5 status\_t FTM\_SetupPwm ( *FTM\_Type* \* *base*, *const ftm\_chnl\_pwm\_signal\_param\_t* \* *chnlParams*, *uint8\_t* *numOfChnls*, *ftm\_pwm\_mode\_t* *mode*, *uint32\_t* *pwmFreq\_Hz*, *uint32\_t* *srcClock\_Hz* )**

Call this function to configure the PWM signal period, mode, duty cycle, and edge. Use this function to configure all FTM channels that are used to output a PWM signal.

Parameters

|                    |                                                                                     |
|--------------------|-------------------------------------------------------------------------------------|
| <i>base</i>        | FTM peripheral base address                                                         |
| <i>chnlParams</i>  | Array of PWM channel parameters to configure the channel(s)                         |
| <i>numOfChnls</i>  | Number of channels to configure; This should be the size of the array passed in     |
| <i>mode</i>        | PWM operation mode, options available in enumeration <a href="#">ftm_pwm_mode_t</a> |
| <i>pwmFreq_Hz</i>  | PWM signal frequency in Hz                                                          |
| <i>srcClock_Hz</i> | FTM counter clock in Hz                                                             |

Returns

kStatus\_Success if the PWM setup was successful kStatus\_Error on failure

#### 16.8.6 status\_t **FTM\_UpdatePwmDutycycle** ( **FTM\_Type** \* *base*, **ftm\_chnl\_t** *chnlNumber*, **ftm\_pwm\_mode\_t** *currentPwmMode*, **uint8\_t** *dutyCyclePercent* )

Parameters

|                         |                                                                                                                                   |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>             | FTM peripheral base address                                                                                                       |
| <i>chnlNumber</i>       | The channel/channel pair number. In combined mode, this represents the channel pair number                                        |
| <i>currentPwmMode</i>   | The current PWM mode set during PWM setup                                                                                         |
| <i>dutyCyclePercent</i> | New PWM pulse width; The value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle) |

Returns

kStatus\_Success if the PWM update was successful kStatus\_Error on failure

#### 16.8.7 void **FTM\_UpdateChnlEdgeLevelSelect** ( **FTM\_Type** \* *base*, **ftm\_chnl\_t** *chnlNumber*, **uint8\_t** *level* )

Parameters

|                   |                                                                                                                                                   |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | FTM peripheral base address                                                                                                                       |
| <i>chnlNumber</i> | The channel number                                                                                                                                |
| <i>level</i>      | The level to be set to the ELSnB:ELSnA field; Valid values are 00, 01, 10, 11. See the Kinetis SoC reference manual for details about this field. |

#### 16.8.8 **status\_t FTM\_SetupPwmMode ( FTM\_Type \* *base*, const ftm\_chnl\_pwm\_config\_param\_t \* *chnlParams*, uint8\_t *numOfChnls*, ftm\_pwm\_mode\_t *mode* )**

Call this function to configure the PWM signal mode, duty cycle in ticks, and edge. Use this function to configure all FTM channels that are used to output a PWM signal. Please note that: This API is similar with [FTM\\_SetupPwm\(\)](#) API, but will not set the timer period, and this API will set channel match value in timer ticks, not period percent.

Parameters

|                   |                                                                                     |
|-------------------|-------------------------------------------------------------------------------------|
| <i>base</i>       | FTM peripheral base address                                                         |
| <i>chnlParams</i> | Array of PWM channel parameters to configure the channel(s)                         |
| <i>numOfChnls</i> | Number of channels to configure; This should be the size of the array passed in     |
| <i>mode</i>       | PWM operation mode, options available in enumeration <a href="#">ftm_pwm_mode_t</a> |

Returns

kStatus\_Success if the PWM setup was successful kStatus\_Error on failure

#### 16.8.9 **void FTM\_SetupInputCapture ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlNumber*, ftm\_input\_capture\_edge\_t *captureMode*, uint32\_t *filterValue* )**

When the edge specified in the captureMode argument occurs on the channel, the FTM counter is captured into the CnV register. The user has to read the CnV register separately to get this value. The filter function is disabled if the filterVal argument passed in is 0. The filter function is available only for channels 0, 1, 2, 3.

Parameters

|                    |                                                                             |
|--------------------|-----------------------------------------------------------------------------|
| <i>base</i>        | FTM peripheral base address                                                 |
| <i>chnlNumber</i>  | The channel number                                                          |
| <i>captureMode</i> | Specifies which edge to capture                                             |
| <i>filterValue</i> | Filter value, specify 0 to disable filter. Available only for channels 0-3. |

#### 16.8.10 void FTM\_SetupOutputCompare ( **FTM\_Type** \* *base*, **ftm\_chnl\_t** *chnlNumber*, **ftm\_output\_compare\_mode\_t** *compareMode*, **uint32\_t** *compareValue* )

When the FTM counter matches the value of compareVal argument (this is written into CnV reg), the channel output is changed based on what is specified in the compareMode argument.

Parameters

|                     |                                                                        |
|---------------------|------------------------------------------------------------------------|
| <i>base</i>         | FTM peripheral base address                                            |
| <i>chnlNumber</i>   | The channel number                                                     |
| <i>compareMode</i>  | Action to take on the channel output when the compare condition is met |
| <i>compareValue</i> | Value to be programmed in the CnV register.                            |

#### 16.8.11 void FTM\_SetupDualEdgeCapture ( **FTM\_Type** \* *base*, **ftm\_chnl\_t** *chnlPairNumber*, **const ftm\_dual\_edge\_capture\_param\_t** \* *edgeParam*, **uint32\_t** *filterValue* )

This function sets up the dual edge capture mode on a channel pair. The capture edge for the channel pair and the capture mode (one-shot or continuous) is specified in the parameter argument. The filter function is disabled if the filterVal argument passed is zero. The filter function is available only on channels 0 and 2. The user has to read the channel CnV registers separately to get the capture values.

Parameters

|                        |                                                     |
|------------------------|-----------------------------------------------------|
| <i>base</i>            | FTM peripheral base address                         |
| <i>chnlPair-Number</i> | The FTM channel pair number; options are 0, 1, 2, 3 |

|                    |                                                                                     |
|--------------------|-------------------------------------------------------------------------------------|
| <i>edgeParam</i>   | Sets up the dual edge capture function                                              |
| <i>filterValue</i> | Filter value, specify 0 to disable filter. Available only for channel pair 0 and 1. |

### 16.8.12 void FTM\_SetupFaultInput ( **FTM\_Type** \* *base*, **ftm\_fault\_input\_t** *faultNumber*, **const ftm\_fault\_param\_t** \* *faultParams* )

FTM can have up to 4 fault inputs. This function sets up fault parameters, fault level, and input filter.

Parameters

|                    |                                          |
|--------------------|------------------------------------------|
| <i>base</i>        | FTM peripheral base address              |
| <i>faultNumber</i> | FTM fault to configure.                  |
| <i>faultParams</i> | Parameters passed in to set up the fault |

### 16.8.13 void FTM\_EnableInterrupts ( **FTM\_Type** \* *base*, **uint32\_t** *mask* )

Parameters

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

### 16.8.14 void FTM\_DisableInterrupts ( **FTM\_Type** \* *base*, **uint32\_t** *mask* )

Parameters

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

### 16.8.15 **uint32\_t** FTM\_GetEnabledInterrupts ( **FTM\_Type** \* *base* )

Parameters

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

Returns

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

### 16.8.16 `uint32_t FTM_GetStatusFlags ( FTM_Type * base )`

Parameters

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

Returns

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

### 16.8.17 `void FTM_ClearStatusFlags ( FTM_Type * base, uint32_t mask )`

Parameters

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

### 16.8.18 `static void FTM_SetTimerPeriod ( FTM_Type * base, uint32_t ticks ) [inline], [static]`

Timers counts from 0 until it equals the count value set here. The count value is written to the MOD register.

Note

1. This API allows the user to use the FTM module as a timer. Do not mix usage of this API with FTM's PWM setup API's.
2. Call the utility macros provided in the fsl\_common.h to convert usec or msec to ticks.

Parameters

|              |                                                                            |
|--------------|----------------------------------------------------------------------------|
| <i>base</i>  | FTM peripheral base address                                                |
| <i>ticks</i> | A timer period in units of ticks, which should be equal or greater than 1. |

### 16.8.19 static uint32\_t FTM\_GetCurrentTimerCount ( **FTM\_Type** \* *base* ) [**inline**], [**static**]

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note

Call the utility macros provided in the fsl\_common.h to convert ticks to usec or msec.

Parameters

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

Returns

The current counter value in ticks

### 16.8.20 static uint32\_t FTM\_GetInputCaptureValue ( **FTM\_Type** \* *base*, **ftm\_chnl\_t chnlNumber** ) [**inline**], [**static**]

This function returns the captured value of a FTM channel configured in input capture or dual edge capture mode.

Note

Call the utility macros provided in the fsl\_common.h to convert ticks to usec or msec.

Parameters

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

|                   |                    |
|-------------------|--------------------|
| <i>chnlNumber</i> | Channel to be read |
|-------------------|--------------------|

Returns

The captured FTM counter value of the input modes.

#### 16.8.21 static void FTM\_StartTimer ( **FTM\_Type** \* *base*, **ftm\_clock\_source\_t** *clockSource* ) [inline], [static]

Parameters

|                    |                                                                              |
|--------------------|------------------------------------------------------------------------------|
| <i>base</i>        | FTM peripheral base address                                                  |
| <i>clockSource</i> | FTM clock source; After the clock source is set, the counter starts running. |

#### 16.8.22 static void FTM\_StopTimer ( **FTM\_Type** \* *base* ) [inline], [static]

Parameters

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

#### 16.8.23 static void FTM\_SetSoftwareCtrlEnable ( **FTM\_Type** \* *base*, **ftm\_chnl\_t** *chnlNumber*, **bool** *value* ) [inline], [static]

Parameters

|                   |                                                                                                                            |
|-------------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | FTM peripheral base address                                                                                                |
| <i>chnlNumber</i> | Channel to be enabled or disabled                                                                                          |
| <i>value</i>      | true: channel output is affected by software output control false: channel output is unaffected by software output control |

#### 16.8.24 static void FTM\_SetSoftwareCtrlVal ( **FTM\_Type** \* *base*, **ftm\_chnl\_t** *chnlNumber*, **bool** *value* ) [inline], [static]

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>base</i>       | FTM peripheral base address.  |
| <i>chnlNumber</i> | Channel to be configured      |
| <i>value</i>      | true to set 1, false to set 0 |

**16.8.25 static void FTM\_SetGlobalTimeBaseOutputEnable ( FTM\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FTM peripheral base address      |
| <i>enable</i> | true to enable, false to disable |

**16.8.26 static void FTM\_SetOutputMask ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlNumber*, bool *mask* ) [inline], [static]**

Parameters

|                   |                                                                        |
|-------------------|------------------------------------------------------------------------|
| <i>base</i>       | FTM peripheral base address                                            |
| <i>chnlNumber</i> | Channel to be configured                                               |
| <i>mask</i>       | true: masked, channel is forced to its inactive state; false: unmasked |

**16.8.27 static void FTM\_SetFaultControlEnable ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlPairNumber*, bool *value* ) [inline], [static]**

Parameters

|                        |                                                     |
|------------------------|-----------------------------------------------------|
| <i>base</i>            | FTM peripheral base address                         |
| <i>chnlPair-Number</i> | The FTM channel pair number; options are 0, 1, 2, 3 |

|              |                                                                           |
|--------------|---------------------------------------------------------------------------|
| <i>value</i> | true: Enable fault control for this channel pair; false: No fault control |
|--------------|---------------------------------------------------------------------------|

**16.8.28 static void FTM\_SetDeadTimeEnable ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlPairNumber*, bool *value* ) [inline], [static]**

Parameters

|                        |                                                                           |
|------------------------|---------------------------------------------------------------------------|
| <i>base</i>            | FTM peripheral base address                                               |
| <i>chnlPair-Number</i> | The FTM channel pair number; options are 0, 1, 2, 3                       |
| <i>value</i>           | true: Insert dead time in this channel pair; false: No dead time inserted |

**16.8.29 static void FTM\_SetComplementaryEnable ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlPairNumber*, bool *value* ) [inline], [static]**

Parameters

|                        |                                                                    |
|------------------------|--------------------------------------------------------------------|
| <i>base</i>            | FTM peripheral base address                                        |
| <i>chnlPair-Number</i> | The FTM channel pair number; options are 0, 1, 2, 3                |
| <i>value</i>           | true: enable complementary mode; false: disable complementary mode |

**16.8.30 static void FTM\_SetInvertEnable ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlPairNumber*, bool *value* ) [inline], [static]**

Parameters

|                        |                                                     |
|------------------------|-----------------------------------------------------|
| <i>base</i>            | FTM peripheral base address                         |
| <i>chnlPair-Number</i> | The FTM channel pair number; options are 0, 1, 2, 3 |

|              |                                                  |
|--------------|--------------------------------------------------|
| <i>value</i> | true: enable inverting; false: disable inverting |
|--------------|--------------------------------------------------|

**16.8.31 void FTM\_SetupQuadDecode ( FTM\_Type \* *base*, const ftm\_phase\_params\_t \* *phaseAParams*, const ftm\_phase\_params\_t \* *phaseBParams*, ftm\_quad\_decode\_mode\_t *quadMode* )**

Parameters

|                     |                                                       |
|---------------------|-------------------------------------------------------|
| <i>base</i>         | FTM peripheral base address                           |
| <i>phaseAParams</i> | Phase A configuration parameters                      |
| <i>phaseBParams</i> | Phase B configuration parameters                      |
| <i>quadMode</i>     | Selects encoding mode used in quadrature decoder mode |

**16.8.32 static uint32\_t FTM\_GetQuadDecoderFlags ( FTM\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

Flag mask of FTM Quad Decoder, see \_ftm\_quad\_decoder\_flags.

**16.8.33 static void FTM\_SetQuadDecoderModuloValue ( FTM\_Type \* *base*, uint32\_t *startValue*, uint32\_t *overValue* ) [inline], [static]**

The modulo values configure the minimum and maximum values that the Quad decoder counter can reach. After the counter goes over, the counter value goes to the other side and decrease/increase again.

Parameters

|                   |                                                |
|-------------------|------------------------------------------------|
| <i>base</i>       | FTM peripheral base address.                   |
| <i>startValue</i> | The low limit value for Quad Decoder counter.  |
| <i>overValue</i>  | The high limit value for Quad Decoder counter. |

**16.8.34 static uint32\_t FTM\_GetQuadDecoderCounterValue ( FTM\_Type \* *base* )  
[inline], [static]**

Parameters

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

Returns

Current quad Decoder counter value.

**16.8.35 static void FTM\_ClearQuadDecoderCounterValue ( FTM\_Type \* *base* )  
[inline], [static]**

The counter is set as the initial value.

Parameters

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

**16.8.36 static void FTM\_SetSoftwareTrigger ( FTM\_Type \* *base*, bool *enable* )  
[inline], [static]**

Parameters

|               |                                                                             |
|---------------|-----------------------------------------------------------------------------|
| <i>base</i>   | FTM peripheral base address                                                 |
| <i>enable</i> | true: software trigger is selected, false: software trigger is not selected |

**16.8.37 static void FTM\_SetWriteProtection ( FTM\_Type \* *base*, bool *enable* )  
[inline], [static]**

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | FTM peripheral base address                                            |
| <i>enable</i> | true: Write-protection is enabled, false: Write-protection is disabled |

### 16.8.38 static void FTM\_EnableDmaTransfer ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlNumber*, bool *enable* ) [inline], [static]

Note: CHnIE bit needs to be set when calling this API. The channel DMA transfer request is generated and the channel interrupt is not generated if (CHnF = 1) when DMA and CHnIE bits are set.

Parameters

|                   |                                  |
|-------------------|----------------------------------|
| <i>base</i>       | FTM peripheral base address.     |
| <i>chnlNumber</i> | Channel to be configured         |
| <i>enable</i>     | true to enable, false to disable |

# Chapter 17

## GPIO: General-Purpose Input/Output Driver

### 17.1 Overview

#### Modules

- FGPIO Driver
- GPIO Driver

#### 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 }  
*GPIO direction definition.*

#### Driver version

- #define `FSL_GPIO_DRIVER_VERSION` (`MAKE_VERSION(2, 6, 0)`)  
*GPIO driver version.*

### 17.2 Data Structure Documentation

#### 17.2.1 struct `gpio_pin_config_t`

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the `outputConfig` unused. Note that in some use cases, the corresponding port property should be configured in advance with the [PORT\\_SetPinConfig\(\)](#).

#### Data Fields

- `gpio_pin_direction_t pinDirection`  
*GPIO direction, input or output.*
- `uint8_t outputLogic`  
*Set a default output logic, which has no use in input.*

## 17.3 Macro Definition Documentation

17.3.1 `#define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))`

## 17.4 Enumeration Type Documentation

17.4.1 `enum gpio_pin_direction_t`

Enumerator

*kGPIO\_DigitalInput* Set current pin as digital input.

*kGPIO\_DigitalOutput* Set current pin as digital output.

## 17.5 GPIO Driver

### 17.5.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General-Purpose Input/Output (GPIO) module of MCUXpresso SDK devices.

### 17.5.2 Typical use case

#### 17.5.2.1 Output Operation

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/gpio

#### 17.5.2.2 Input Operation

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/gpio

## GPIO Configuration

- void [GPIO\\_PinInit](#) (GPIO\_Type \*base, 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 pin, uint8\_t output)  
*Sets the output level of the multiple GPIO pins to the logic 1 or 0.*
- static void [GPIO\\_PortSet](#) (GPIO\_Type \*base, 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 mask)  
*Sets the output level of the multiple GPIO pins to the logic 0.*
- static void [GPIO\\_PortToggle](#) (GPIO\_Type \*base, uint32\_t mask)  
*Reverses the current output logic of the multiple GPIO pins.*

## GPIO Input Operations

- static uint32\_t [GPIO\\_PinRead](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current input value of the GPIO port.*

## GPIO Interrupt

- uint32\_t [GPIO\\_PortGetInterruptFlags](#) (GPIO\_Type \*base)  
*Reads the GPIO port interrupt status flag.*

- void [GPIO\\_PortClearInterruptFlags](#) (GPIO\_Type \*base, uint32\_t mask)  
*Clears multiple GPIO pin interrupt status flags.*

### 17.5.3 Function Documentation

#### 17.5.3.1 void [GPIO\\_PinInit](#) ( **GPIO\_Type** \* *base*, **uint32\_t** *pin*, **const gpio\_pin\_config\_t** \* *config* )

To initialize the GPIO, define a pin configuration, as either input or output, in the user file. Then, call the [GPIO\\_PinInit\(\)](#) function.

This is an example to define an input pin or an output pin configuration.

```
* 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 (GPIOA, GPIOB, GPIOC, and so on.) |
| <i>pin</i>    | GPIO port pin number                                           |
| <i>config</i> | GPIO pin configuration pointer                                 |

#### 17.5.3.2 static void [GPIO\\_PinWrite](#) ( **GPIO\_Type** \* *base*, **uint32\_t** *pin*, **uint8\_t** *output* ) **[inline], [static]**

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.) |
| <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> |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 17.5.3.3 static void GPIO\_PortSet ( **GPIO\_Type** \* *base*, **uint32\_t** *mask* ) [inline], [static]

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.) |
| <i>mask</i> | GPIO pin number macro                                          |

### 17.5.3.4 static void GPIO\_PortClear ( **GPIO\_Type** \* *base*, **uint32\_t** *mask* ) [inline], [static]

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.) |
| <i>mask</i> | GPIO pin number macro                                          |

### 17.5.3.5 static void GPIO\_PortToggle ( **GPIO\_Type** \* *base*, **uint32\_t** *mask* ) [inline], [static]

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.) |
| <i>mask</i> | GPIO pin number macro                                          |

### 17.5.3.6 static **uint32\_t** GPIO\_PinRead ( **GPIO\_Type** \* *base*, **uint32\_t** *pin* ) [inline], [static]

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.) |
| <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> |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

#### 17.5.3.7 `uint32_t GPIO_PortGetInterruptFlags ( GPIO_Type * base )`

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.) |
|-------------|----------------------------------------------------------------|

Return values

|            |                                                                                                             |
|------------|-------------------------------------------------------------------------------------------------------------|
| <i>The</i> | current GPIO port interrupt status flag, for example, 0x00010001 means the pin 0 and 17 have the interrupt. |
|------------|-------------------------------------------------------------------------------------------------------------|

#### 17.5.3.8 `void GPIO_PortClearInterruptFlags ( GPIO_Type * base, uint32_t mask )`

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.) |
| <i>mask</i> | GPIO pin number macro                                          |

## 17.6 FGPIO Driver

This section describes the programming interface of the FGPIO driver. The FGPIO driver configures the FGPIO module and provides a functional interface to build the GPIO application.

Note

FGPIO (Fast GPIO) is only available in a few MCUs. FGPIO and GPIO share the same peripheral but use different registers. FGPIO is closer to the core than the regular GPIO and it's faster to read and write.

### 17.6.1 Typical use case

#### 17.6.1.1 Output Operation

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/gpio

#### 17.6.1.2 Input Operation

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/gpio

# Chapter 18

## I2C: Inter-Integrated Circuit Driver

### 18.1 Overview

#### Modules

- I2C CMSIS Driver
- I2C Driver
- I2C FreeRTOS Driver
- I2C eDMA Driver

## 18.2 I2C Driver

### 18.2.1 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 target the 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 knowing 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 target the 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.

### 18.2.2 Typical use case

#### 18.2.2.1 Master Operation in functional method

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/i2c

#### 18.2.2.2 Master Operation in interrupt transactional method

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/i2c

#### 18.2.2.3 Master Operation in DMA transactional method

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/i2c

#### 18.2.2.4 Slave Operation in functional method

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/i2c

#### 18.2.2.5 Slave Operation in interrupt transactional method

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/i2c

## Data Structures

- struct `i2c_master_config_t`  
*I2C master user configuration.* [More...](#)
- struct `i2c_slave_config_t`  
*I2C slave user configuration.* [More...](#)
- struct `i2c_master_transfer_t`  
*I2C master transfer structure.* [More...](#)
- struct `i2c_master_handle_t`  
*I2C master handle structure.* [More...](#)
- struct `i2c_slave_transfer_t`  
*I2C slave transfer structure.* [More...](#)
- struct `i2c_slave_handle_t`  
*I2C slave handle structure.* [More...](#)

## Macros

- #define `I2C_RETRY_TIMES` 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/  
*Retry times for waiting flag.*
- #define `I2C_MASTER_FACK_CONTROL` 0U /\* Default defines to zero means master will send ack automatically. \*/  
*Mater Fast ack control, control if master needs to manually write ack, this is used to low the speed of transfer for SoCs with feature FSL\_FEATURE\_I2C\_HAS\_DOUBLE\_BUFFERING.*

## Typedefs

- typedef void(\* `i2c_master_transfer_callback_t` )(I2C\_Type \*base, `i2c_master_handle_t` \*handle, `status_t` status, void \*userData)  
*I2C master transfer callback typedef.*
- typedef void(\* `i2c_slave_transfer_callback_t` )(I2C\_Type \*base, `i2c_slave_transfer_t` \*xfer, void \*userData)  
*I2C slave transfer callback typedef.*

## Enumerations

- enum {
 `kStatus_I2C_Busy` = MAKE\_STATUS(kStatusGroup\_I2C, 0),
 `kStatus_I2C_Idle` = MAKE\_STATUS(kStatusGroup\_I2C, 1),
 `kStatus_I2C_Nak` = MAKE\_STATUS(kStatusGroup\_I2C, 2),
 `kStatus_I2C_ArbitrationLost` = MAKE\_STATUS(kStatusGroup\_I2C, 3),
 `kStatus_I2C_Timeout` = MAKE\_STATUS(kStatusGroup\_I2C, 4),
 `kStatus_I2C_Addr_Nak` = MAKE\_STATUS(kStatusGroup\_I2C, 5) }
   
*I2C status return codes.*

- enum `_i2c_flags` {
   
  `kI2C_ReceiveNakFlag` = I2C\_S\_RXAK\_MASK,
   
  `kI2C_IntPendingFlag` = I2C\_S\_IICIF\_MASK,
   
  `kI2C_TransferDirectionFlag` = I2C\_S\_SRW\_MASK,
   
  `kI2C_RangeAddressMatchFlag` = I2C\_S\_RAM\_MASK,
   
  `kI2C_ArbitrationLostFlag` = I2C\_S\_ARBL\_MASK,
   
  `kI2C_BusBusyFlag` = I2C\_S\_BUSY\_MASK,
   
  `kI2C_AddressMatchFlag` = I2C\_S\_IAAS\_MASK,
   
  `kI2C_TransferCompleteFlag` = I2C\_S\_TCF\_MASK,
   
  `kI2C_StopDetectFlag` = I2C\_FLT\_STOPF\_MASK << 8,
   
  `kI2C_StartDetectFlag` = I2C\_FLT\_STARTF\_MASK << 8 }
   
    *I2C peripheral flags.*
- enum `_i2c_interrupt_enable` {
   
  `kI2C_GlobalInterruptEnable` = I2C\_C1\_IICIE\_MASK,
   
  `kI2C_StartStopDetectInterruptEnable` = I2C\_FLT\_SSIE\_MASK }
- enum `i2c_direction_t` {
   
  `kI2C_Write` = 0x0U,
   
  `kI2C_Read` = 0x1U }
   
    *The direction of master and slave transfers.*
- enum `i2c_slave_address_mode_t` {
   
  `kI2C_Address7bit` = 0x0U,
   
  `kI2C_RangeMatch` = 0X2U }
   
    *Addressing mode.*
- enum `_i2c_master_transfer_flags` {
   
  `kI2C_TransferDefaultFlag` = 0x0U,
   
  `kI2C_TransferNoStartFlag` = 0x1U,
   
  `kI2C_TransferRepeatedStartFlag` = 0x2U,
   
  `kI2C_TransferNoStopFlag` = 0x4U }
   
    *I2C transfer control flag.*
- enum `i2c_slave_transfer_event_t` {
   
  `kI2C_SlaveAddressMatchEvent` = 0x01U,
   
  `kI2C_SlaveTransmitEvent` = 0x02U,
   
  `kI2C_SlaveReceiveEvent` = 0x04U,
   
  `kI2C_SlaveTransmitAckEvent` = 0x08U,
   
  `kI2C_SlaveStartEvent` = 0x10U,
   
  `kI2C_SlaveCompletionEvent` = 0x20U,
   
  `kI2C_SlaveGenaralcallEvent` = 0x40U,
   
  `kI2C_SlaveAllEvents` }
   
    *Set of events sent to the callback for nonblocking slave transfers.*
- enum { `kClearFlags` = `kI2C_ArbitrationLostFlag` | `kI2C_IntPendingFlag` | `kI2C_StartDetectFlag` | `kI2C_StopDetectFlag` }
   
    *Common sets of flags used by the driver.*

## Driver version

- #define `FSL_I2C_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 9)`)  
*I2C driver version.*

## Initialization and deinitialization

- void `I2C_MasterInit` (I2C\_Type \*base, const `i2c_master_config_t` \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes the I2C peripheral.*
- void `I2C_SlaveInit` (I2C\_Type \*base, const `i2c_slave_config_t` \*slaveConfig, uint32\_t srcClock\_Hz)  
*Initializes the I2C peripheral.*
- void `I2C_MasterDeinit` (I2C\_Type \*base)  
*De-initializes the I2C master peripheral.*
- void `I2C_SlaveDeinit` (I2C\_Type \*base)  
*De-initializes the I2C slave peripheral.*
- uint32\_t `I2CGetInstance` (I2C\_Type \*base)  
*Get instance number for I2C module.*
- void `I2C_MasterGetDefaultConfig` (`i2c_master_config_t` \*masterConfig)  
*Sets the I2C master configuration structure to default values.*
- void `I2C_SlaveGetDefaultConfig` (`i2c_slave_config_t` \*slaveConfig)  
*Sets the I2C slave configuration structure to default values.*
- static void `I2C_Enable` (I2C\_Type \*base, bool enable)  
*Enables or disables the I2C peripheral operation.*

## Status

- uint32\_t `I2C_MasterGetStatusFlags` (I2C\_Type \*base)  
*Gets the I2C status flags.*
- static uint32\_t `I2C_SlaveGetStatusFlags` (I2C\_Type \*base)  
*Gets the I2C status flags.*
- static void `I2C_MasterClearStatusFlags` (I2C\_Type \*base, uint32\_t statusMask)  
*Clears the I2C status flag state.*
- static void `I2C_SlaveClearStatusFlags` (I2C\_Type \*base, uint32\_t statusMask)  
*Clears the I2C status flag state.*

## Interrupts

- void `I2C_EnableInterrupts` (I2C\_Type \*base, uint32\_t mask)  
*Enables I2C interrupt requests.*
- void `I2C_DisableInterrupts` (I2C\_Type \*base, uint32\_t mask)  
*Disables I2C interrupt requests.*

## DMA Control

- static void [I2C\\_EnableDMA](#) (I2C\_Type \*base, bool enable)  
*Enables/disables the I2C DMA interrupt.*
- static uint32\_t [I2C\\_GetDataRegAddr](#) (I2C\_Type \*base)  
*Gets the I2C tx/rx data register address.*

## Bus Operations

- void [I2C\\_MasterSetBaudRate](#) (I2C\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the I2C master transfer baud rate.*
- 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.*
- 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 uint8\_t \*txBuff, size\_t txSize, uint32\_t flags)  
*Performs a polling send transaction on the I2C bus.*
- status\_t [I2C\\_MasterReadBlocking](#) (I2C\_Type \*base, uint8\_t \*rxBuff, size\_t rxSize, uint32\_t flags)  
*Performs a polling receive transaction on the I2C bus.*
- status\_t [I2C\\_SlaveWriteBlocking](#) (I2C\_Type \*base, const uint8\_t \*txBuff, size\_t txSize)  
*Performs a polling send transaction on the I2C bus.*
- status\_t [I2C\\_SlaveReadBlocking](#) (I2C\_Type \*base, uint8\_t \*rxBuff, size\_t rxSize)  
*Performs a polling receive transaction 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.*

## Transactional

- void [I2C\\_MasterTransferCreateHandle](#) (I2C\_Type \*base, [i2c\\_master\\_handle\\_t](#) \*handle, [i2c\\_master\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the I2C handle which is used in transactional functions.*
- status\_t [I2C\\_MasterTransferNonBlocking](#) (I2C\_Type \*base, [i2c\\_master\\_handle\\_t](#) \*handle, [i2c\\_master\\_transfer\\_t](#) \*xfer)  
*Performs a master interrupt non-blocking transfer on the I2C bus.*
- status\_t [I2C\\_MasterTransferGetCount](#) (I2C\_Type \*base, [i2c\\_master\\_handle\\_t](#) \*handle, size\_t \*count)  
*Gets the master transfer status during a interrupt non-blocking transfer.*
- status\_t [I2C\\_MasterTransferAbort](#) (I2C\_Type \*base, [i2c\\_master\\_handle\\_t](#) \*handle)  
*Aborts an interrupt non-blocking transfer early.*
- void [I2C\\_MasterTransferHandleIRQ](#) (I2C\_Type \*base, void \*i2cHandle)  
*Master interrupt handler.*
- void [I2C\\_SlaveTransferCreateHandle](#) (I2C\_Type \*base, [i2c\\_slave\\_handle\\_t](#) \*handle, [i2c\\_slave\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the I2C handle which is used in transactional functions.*

- **status\_t I2C\_SlaveTransferNonBlocking** (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle, uint32\_t eventMask)
 

*Starts accepting slave transfers.*
- **void I2C\_SlaveTransferAbort** (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle)
 

*Aborts the slave transfer.*
- **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.*
- **void I2C\_SlaveTransferHandleIRQ** (I2C\_Type \*base, void \*i2cHandle)
 

*Slave interrupt handler.*

## 18.2.3 Data Structure Documentation

### 18.2.3.1 struct i2c\_master\_config\_t

#### Data Fields

- **bool enableMaster**

*Enables the I2C peripheral at initialization time.*
- **bool enableStopHold**

*Controls the stop hold enable.*
- **uint32\_t baudRate\_Bps**

*Baud rate configuration of I2C peripheral.*
- **uint8\_t glitchFilterWidth**

*Controls the width of the glitch.*

#### Field Documentation

- (1) **bool i2c\_master\_config\_t::enableMaster**
- (2) **bool i2c\_master\_config\_t::enableStopHold**
- (3) **uint32\_t i2c\_master\_config\_t::baudRate\_Bps**
- (4) **uint8\_t i2c\_master\_config\_t::glitchFilterWidth**

### 18.2.3.2 struct i2c\_slave\_config\_t

#### Data Fields

- **bool enableSlave**

*Enables the I2C peripheral at initialization time.*
- **bool enableGeneralCall**

*Enables the general call addressing mode.*
- **bool enableWakeUp**

*Enables/disables waking up MCU from low-power mode.*
- **bool enableBaudRateCtl**

*Enables/disables independent slave baud rate on SCL in very fast I2C modes.*
- **uint16\_t slaveAddress**

*A slave address configuration.*

- **uint16\_t upperAddress**  
A maximum boundary slave address used in a range matching mode.
- **i2c\_slave\_address\_mode\_t addressingMode**  
An addressing mode configuration of i2c\_slave\_address\_mode\_config\_t.
- **uint32\_t sclStopHoldTime\_ns**  
the delay from the rising edge of SCL (I2C clock) to the rising edge of SDA (I2C data) while SCL is high (stop condition), SDA hold time and SCL start hold time are also configured according to the SCL stop hold time.

### Field Documentation

- (1) **bool i2c\_slave\_config\_t::enableSlave**
- (2) **bool i2c\_slave\_config\_t::enableGeneralCall**
- (3) **bool i2c\_slave\_config\_t::enableWakeUp**
- (4) **bool i2c\_slave\_config\_t::enableBaudRateCtl**
- (5) **uint16\_t i2c\_slave\_config\_t::slaveAddress**
- (6) **uint16\_t i2c\_slave\_config\_t::upperAddress**
- (7) **i2c\_slave\_address\_mode\_t i2c\_slave\_config\_t::addressingMode**
- (8) **uint32\_t i2c\_slave\_config\_t::sclStopHoldTime\_ns**

### 18.2.3.3 struct i2c\_master\_transfer\_t

#### Data Fields

- **uint32\_t flags**  
A transfer flag which controls the transfer.
- **uint8\_t slaveAddress**  
7-bit slave address.
- **i2c\_direction\_t direction**  
A transfer direction, read or write.
- **uint32\_t subaddress**  
A sub address.
- **uint8\_t subaddressSize**  
A size of the command buffer.
- **uint8\_t \*volatile data**  
A transfer buffer.
- **volatile size\_t dataSize**  
A transfer size.

### Field Documentation

- (1) **uint32\_t i2c\_master\_transfer\_t::flags**

- (2) `uint8_t i2c_master_transfer_t::slaveAddress`
- (3) `i2c_direction_t i2c_master_transfer_t::direction`
- (4) `uint32_t i2c_master_transfer_t::subaddress`

Transferred MSB first.

- (5) `uint8_t i2c_master_transfer_t::subaddressSize`
- (6) `uint8_t* volatile i2c_master_transfer_t::data`
- (7) `volatile size_t i2c_master_transfer_t::dataSize`

#### 18.2.3.4 struct \_i2c\_master\_handle

I2C master handle typedef.

#### Data Fields

- `i2c_master_transfer_t transfer`  
*I2C master transfer copy.*
- `size_t transferSize`  
*Total bytes to be transferred.*
- `uint8_t state`  
*A transfer state maintained during transfer.*
- `i2c_master_transfer_callback_t completionCallback`  
*A callback function called when the transfer is finished.*
- `void *userData`  
*A callback parameter passed to the callback function.*

#### Field Documentation

- (1) `i2c_master_transfer_t i2c_master_handle_t::transfer`
- (2) `size_t i2c_master_handle_t::transferSize`
- (3) `uint8_t i2c_master_handle_t::state`
- (4) `i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback`
- (5) `void* i2c_master_handle_t::userData`

#### 18.2.3.5 struct i2c\_slave\_transfer\_t

#### Data Fields

- `i2c_slave_transfer_event_t event`  
*A reason that the callback is invoked.*
- `uint8_t *volatile data`

- **volatile size\_t dataSize**  
*A transfer buffer.*
- **status\_t completionStatus**  
*A transfer size.*
- **status\_t completionStatus**  
*Success or error code describing how the transfer completed.*
- **size\_t transferredCount**  
*A number of bytes actually transferred since the start or since the last repeated start.*

### Field Documentation

- (1) **i2c\_slave\_transfer\_event\_t i2c\_slave\_transfer\_t::event**
  - (2) **uint8\_t\* volatile i2c\_slave\_transfer\_t::data**
  - (3) **volatile size\_t i2c\_slave\_transfer\_t::dataSize**
  - (4) **status\_t i2c\_slave\_transfer\_t::completionStatus**
  - (5) **size\_t i2c\_slave\_transfer\_t::transferredCount**
- Only applies for [kI2C\\_SlaveCompletionEvent](#).

- (5) **size\_t i2c\_slave\_transfer\_t::transferredCount**

### 18.2.3.6 struct \_i2c\_slave\_handle

I2C slave handle typedef.

### Data Fields

- **volatile bool isBusy**  
*Indicates whether a transfer is busy.*
- **i2c\_slave\_transfer\_t transfer**  
*I2C slave transfer copy.*
- **uint32\_t eventMask**  
*A mask of enabled events.*
- **i2c\_slave\_transfer\_callback\_t callback**  
*A callback function called at the transfer event.*
- **void \* userData**  
*A callback parameter passed to the callback.*

### Field Documentation

- (1) **volatile bool i2c\_slave\_handle\_t::isBusy**
- (2) **i2c\_slave\_transfer\_t i2c\_slave\_handle\_t::transfer**
- (3) **uint32\_t i2c\_slave\_handle\_t::eventMask**
- (4) **i2c\_slave\_transfer\_callback\_t i2c\_slave\_handle\_t::callback**
- (5) **void\* i2c\_slave\_handle\_t::userData**

## 18.2.4 Macro Definition Documentation

**18.2.4.1 #define FSL\_I2C\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 9))**

**18.2.4.2 #define I2C\_RETRY\_TIMES 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/**

## 18.2.5 Typedef Documentation

**18.2.5.1 typedef void(\* i2c\_master\_transfer\_callback\_t)(I2C\_Type \*base, i2c\_master\_handle\_t \*handle, status\_t status, void \*userData)**

**18.2.5.2 typedef void(\* i2c\_slave\_transfer\_callback\_t)(I2C\_Type \*base, i2c\_slave\_transfer\_t \*xfer, void \*userData)**

## 18.2.6 Enumeration Type Documentation

### 18.2.6.1 anonymous enum

Enumerator

*kStatus\_I2C\_Busy* I2C is busy with current transfer.

*kStatus\_I2C\_Idle* Bus is Idle.

*kStatus\_I2C\_Nak* NAK received during transfer.

*kStatus\_I2C\_ArbitrationLost* Arbitration lost during transfer.

*kStatus\_I2C\_Timeout* Timeout polling status flags.

*kStatus\_I2C\_Addr\_Nak* NAK received during the address probe.

### 18.2.6.2 enum \_i2c\_flags

Note

These enumerations are meant to be OR'd together to form a bit mask.

Enumerator

*kI2C\_ReceiveNakFlag* I2C receive NAK flag.

*kI2C\_IntPendingFlag* I2C interrupt pending flag. This flag can be cleared.

*kI2C\_TransferDirectionFlag* I2C transfer direction flag.

*kI2C\_RangeAddressMatchFlag* I2C range address match flag.

*kI2C\_ArbitrationLostFlag* I2C arbitration lost flag. This flag can be cleared.

*kI2C\_BusBusyFlag* I2C bus busy flag.

*kI2C\_AddressMatchFlag* I2C address match flag.

*kI2C\_TransferCompleteFlag* I2C transfer complete flag.

***kI2C\_StopDetectFlag*** I2C stop detect flag. This flag can be cleared.  
***kI2C\_StartDetectFlag*** I2C start detect flag. This flag can be cleared.

### 18.2.6.3 enum \_i2c\_interrupt\_enable

Enumerator

***kI2C\_GlobalInterruptEnable*** I2C global interrupt.  
***kI2C\_StartStopDetectInterruptEnable*** I2C start&stop detect interrupt.

### 18.2.6.4 enum i2c\_direction\_t

Enumerator

***kI2C\_Write*** Master transmits to the slave.  
***kI2C\_Read*** Master receives from the slave.

### 18.2.6.5 enum i2c\_slave\_address\_mode\_t

Enumerator

***kI2C\_Address7bit*** 7-bit addressing mode.  
***kI2C\_RangeMatch*** Range address match addressing mode.

### 18.2.6.6 enum \_i2c\_master\_transfer\_flags

Enumerator

***kI2C\_TransferDefaultFlag*** A transfer starts with a start signal, stops with a stop signal.  
***kI2C\_TransferNoStartFlag*** A transfer starts without a start signal, only support write only or write+read with no start flag, do not support read only with no start flag.  
***kI2C\_TransferRepeatedStartFlag*** A transfer starts with a repeated start signal.  
***kI2C\_TransferNoStopFlag*** A transfer ends without a stop signal.

### 18.2.6.7 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\(\)](#) 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*** A callback is requested to provide data to transmit (slave-transmitter role).

***kI2C\_SlaveReceiveEvent*** A callback is requested to provide a buffer in which to place received data (slave-receiver role).

***kI2C\_SlaveTransmitAckEvent*** A callback needs to either transmit an ACK or NACK.

***kI2C\_SlaveStartEvent*** A start/repeated start was detected.

***kI2C\_SlaveCompletionEvent*** A stop was detected or finished transfer, completing the transfer.

***kI2C\_SlaveGeneralCallEvent*** Received the general call address after a start or repeated start.

***kI2C\_SlaveAllEvents*** A bit mask of all available events.

**18.2.6.8 anonymous enum**

## Enumerator

***kClearFlags*** All flags which are cleared by the driver upon starting a transfer.

**18.2.7 Function Documentation****18.2.7.1 void I2C\_MasterInit ( *I2C\_Type \* base*, *const i2c\_master\_config\_t \* masterConfig*, *uint32\_t srcClock\_Hz* )**

Call this API to ungate the I2C clock and configure the I2C with master configuration.

## Note

This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can be custom filled or it can be set with default values by using the [I2C\\_MasterGetDefaultConfig\(\)](#). After calling this API, the master is ready to transfer. This is an example.

```
* i2c_master_config_t config = {
* .enableMaster = true,
* .enableStopHold = false,
* .highDrive = false,
* .baudRate_Bps = 100000,
* .glitchFilterWidth = 0
* };
* I2C_MasterInit(I2C0, &config, 12000000U);
*
```

## Parameters

|                     |                                                 |
|---------------------|-------------------------------------------------|
| <i>base</i>         | I2C base pointer                                |
| <i>masterConfig</i> | A pointer to the master configuration structure |
| <i>srcClock_Hz</i>  | I2C peripheral clock frequency in Hz            |

**18.2.7.2 void I2C\_SlaveInit ( I2C\_Type \* *base*, const i2c\_slave\_config\_t \* *slaveConfig*, uint32\_t *srcClock\_Hz* )**

Call this API to ungate the I2C clock and initialize the I2C with the slave configuration.

## Note

This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can partly be set with default values by [I2C\\_SlaveGetDefaultConfig\(\)](#) or it can be custom filled by the user. This is an example.

```
* i2c_slave_config_t config = {
* .enableSlave = true,
* .enableGeneralCall = false,
* .addressingMode = kI2C_Address7bit,
* .slaveAddress = 0x1DU,
* .enableWakeUp = false,
* .enableHighDrive = false,
* .enableBaudRateCtl = false,
* .sclStopHoldTime_ns = 4000
* };
* I2C_SlaveInit(I2C0, &config, 12000000U);
*
```

## Parameters

|                    |                                                |
|--------------------|------------------------------------------------|
| <i>base</i>        | I2C base pointer                               |
| <i>slaveConfig</i> | A pointer to the slave configuration structure |
| <i>srcClock_Hz</i> | I2C peripheral clock frequency in Hz           |

**18.2.7.3 void I2C\_MasterDeinit ( I2C\_Type \* *base* )**

Call this API to gate the I2C clock. The I2C master module can't work unless the I2C\_MasterInit is called.

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | I2C base pointer |
|-------------|------------------|

#### 18.2.7.4 void I2C\_SlaveDeinit ( I2C\_Type \* *base* )

Calling this API gates the I2C clock. The I2C slave module can't work unless the I2C\_SlaveInit is called to enable the clock.

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | I2C base pointer |
|-------------|------------------|

#### 18.2.7.5 uint32\_t I2CGetInstance ( I2C\_Type \* *base* )

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | I2C peripheral base address. |
|-------------|------------------------------|

#### 18.2.7.6 void I2C\_MasterGetDefaultConfig ( i2c\_master\_config\_t \* *masterConfig* )

The purpose of this API is to get the configuration structure initialized for use in the I2C\_MasterConfigure(). Use the initialized structure unchanged in the I2C\_MasterConfigure() or modify the structure before calling the I2C\_MasterConfigure(). This is an example.

```
* i2c_master_config_t config;
* I2C_MasterGetDefaultConfig(&config);
*
```

Parameters

|                     |                                                  |
|---------------------|--------------------------------------------------|
| <i>masterConfig</i> | A pointer to the master configuration structure. |
|---------------------|--------------------------------------------------|

#### 18.2.7.7 void I2C\_SlaveGetDefaultConfig ( i2c\_slave\_config\_t \* *slaveConfig* )

The purpose of this API is to get the configuration structure initialized for use in the I2C\_SlaveConfigure(). Modify fields of the structure before calling the I2C\_SlaveConfigure(). This is an example.

```
* i2c_slave_config_t config;
* I2C_SlaveGetDefaultConfig(&config);
*
```

Parameters

|                    |                                                 |
|--------------------|-------------------------------------------------|
| <i>slaveConfig</i> | A pointer to the slave configuration structure. |
|--------------------|-------------------------------------------------|

#### 18.2.7.8 static void I2C\_Enable ( I2C\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>base</i>   | I2C base pointer                                     |
| <i>enable</i> | Pass true to enable and false to disable the module. |

#### 18.2.7.9 uint32\_t I2C\_MasterGetStatusFlags ( I2C\_Type \* *base* )

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | I2C base pointer |
|-------------|------------------|

Returns

status flag, use status flag to AND [\\_i2c\\_flags](#) to get the related status.

#### 18.2.7.10 static uint32\_t I2C\_SlaveGetStatusFlags ( I2C\_Type \* *base* ) [inline], [static]

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | I2C base pointer |
|-------------|------------------|

Returns

status flag, use status flag to AND [\\_i2c\\_flags](#) to get the related status.

#### 18.2.7.11 static void I2C\_MasterClearStatusFlags ( I2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The following status register flags can be cleared kI2C\_ArbitrationLostFlag and kI2C\_IntPendingFlag.

Parameters

|                   |                                                                                                                                                                                                                                                                                                                                            |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | I2C base pointer                                                                                                                                                                                                                                                                                                                           |
| <i>statusMask</i> | <p>The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> <li>• kI2C_StartDetectFlag (if available)</li> <li>• kI2C_StopDetectFlag (if available)</li> <li>• kI2C_ArbitrationLostFlag</li> <li>• kI2C_IntPendingFlagFlag</li> </ul> |

#### 18.2.7.12 static void I2C\_SlaveClearStatusFlags ( I2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The following status register flags can be cleared kI2C\_ArbitrationLostFlag and kI2C\_IntPendingFlag

Parameters

|                   |                                                                                                                                                                                                                                                                                                                                            |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | I2C base pointer                                                                                                                                                                                                                                                                                                                           |
| <i>statusMask</i> | <p>The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> <li>• kI2C_StartDetectFlag (if available)</li> <li>• kI2C_StopDetectFlag (if available)</li> <li>• kI2C_ArbitrationLostFlag</li> <li>• kI2C_IntPendingFlagFlag</li> </ul> |

#### 18.2.7.13 void I2C\_EnableInterrupts ( I2C\_Type \* *base*, uint32\_t *mask* )

Parameters

|             |                                                                                                                                                                                                                                                                                                 |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | I2C base pointer                                                                                                                                                                                                                                                                                |
| <i>mask</i> | <p>interrupt source The parameter can be combination of the following source if defined:</p> <ul style="list-style-type: none"> <li>• kI2C_GlobalInterruptEnable</li> <li>• kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable</li> <li>• kI2C_SdaTimeoutInterruptEnable</li> </ul> |

#### 18.2.7.14 void I2C\_DisableInterrupts ( I2C\_Type \* *base*, uint32\_t *mask* )

Parameters

|             |                                                                                                                                                                                                                                                                                      |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | I2C base pointer                                                                                                                                                                                                                                                                     |
| <i>mask</i> | interrupt source The parameter can be combination of the following source if defined: <ul style="list-style-type: none"><li>• kI2C_GlobalInterruptEnable</li><li>• kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable</li><li>• kI2C_SdaTimeoutInterruptEnable</li></ul> |

#### **18.2.7.15 static void I2C\_EnableDMA ( I2C\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | I2C base pointer                 |
| <i>enable</i> | true to enable, false to disable |

#### **18.2.7.16 static uint32\_t I2C\_GetDataRegAddr ( I2C\_Type \* *base* ) [inline], [static]**

This API is used to provide a transfer address for I2C DMA transfer configuration.

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | I2C base pointer |
|-------------|------------------|

Returns

data register address

#### **18.2.7.17 void I2C\_MasterSetBaudRate ( I2C\_Type \* *base*, uint32\_t *baudRate\_Bps*, uint32\_t *srcClock\_Hz* )**

Parameters

|                     |                            |
|---------------------|----------------------------|
| <i>base</i>         | I2C base pointer           |
| <i>baudRate_Bps</i> | the baud rate value in bps |
| <i>srcClock_Hz</i>  | Source clock               |

### 18.2.7.18 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.                |

### 18.2.7.19 status\_t I2C\_MasterStop ( I2C\_Type \* *base* )

Return values

|                            |                                    |
|----------------------------|------------------------------------|
| <i>kStatus_Success</i>     | Successfully send the stop signal. |
| <i>kStatus_I2C_Timeout</i> | Send stop signal failed, timeout.  |

### 18.2.7.20 status\_t I2C\_MasterRepeatedStart ( I2C\_Type \* *base*, uint8\_t *address*, i2c\_direction\_t *direction* )

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. |

#### 18.2.7.21 **status\_t I2C\_MasterWriteBlocking ( I2C\_Type \* *base*, const uint8\_t \* *txBuff*, size\_t *txSize*, uint32\_t *flags* )**

Parameters

|               |                                                                                                                                                                     |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | The I2C peripheral base pointer.                                                                                                                                    |
| <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 decide whether need to send a stop, use <i>kI2C_TransferDefaultFlag</i> to issue a stop and <i>kI2C_TransferNoStop</i> to not send a stop. |

Return values

|                                     |                                              |
|-------------------------------------|----------------------------------------------|
| <i>kStatus_Success</i>              | Successfully complete the data transmission. |
| <i>kStatus_I2C_Arbitration-Lost</i> | Transfer error, arbitration lost.            |
| <i>kStatus_I2C_Nak</i>              | Transfer error, receive NAK during transfer. |

#### 18.2.7.22 **status\_t I2C\_MasterReadBlocking ( I2C\_Type \* *base*, uint8\_t \* *rxBuff*, size\_t *rxSize*, uint32\_t *flags* )**

Note

The I2C\_MasterReadBlocking function stops the bus before reading the final byte. Without stopping the bus prior for the final read, the bus issues another read, resulting in garbage data being read into the data register.

Parameters

|               |                                                                                                                                                                     |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | I2C peripheral base pointer.                                                                                                                                        |
| <i>rxBuff</i> | The pointer to the data to store the received data.                                                                                                                 |
| <i>rxSize</i> | The length in bytes of the data to be received.                                                                                                                     |
| <i>flags</i>  | Transfer control flag to decide whether need to send a stop, use <i>kI2C_TransferDefaultFlag</i> to issue a stop and <i>kI2C_TransferNoStop</i> to not send a stop. |

Return values

|                            |                                              |
|----------------------------|----------------------------------------------|
| <i>kStatus_Success</i>     | Successfully complete the data transmission. |
| <i>kStatus_I2C_Timeout</i> | Send stop signal failed, timeout.            |

#### 18.2.7.23 **status\_t I2C\_SlaveWriteBlocking ( I2C\_Type \* *base*, const uint8\_t \* *txBuff*, size\_t *txSize* )**

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | The I2C peripheral base pointer.                   |
| <i>txBuff</i> | The pointer to the data to be transferred.         |
| <i>txSize</i> | The length in bytes of the data to be transferred. |

Return values

|                                    |                                              |
|------------------------------------|----------------------------------------------|
| <i>kStatus_Success</i>             | Successfully complete the data transmission. |
| <i>kStatus_I2C_ArbitrationLost</i> | Transfer error, arbitration lost.            |
| <i>kStatus_I2C_Nak</i>             | Transfer error, receive NAK during transfer. |

#### 18.2.7.24 **status\_t I2C\_SlaveReadBlocking ( I2C\_Type \* *base*, uint8\_t \* *rxBuff*, size\_t *rxSize* )**

Parameters

|               |                                                     |
|---------------|-----------------------------------------------------|
| <i>base</i>   | I2C peripheral base pointer.                        |
| <i>rxBuff</i> | The pointer to the data to store the received data. |
| <i>rxSize</i> | The length in bytes of the data to be received.     |

Return values

|                            |                                     |
|----------------------------|-------------------------------------|
| <i>kStatus_Success</i>     | Successfully complete data receive. |
| <i>kStatus_I2C_Timeout</i> | Wait status flag timeout.           |

#### 18.2.7.25 **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. |

## 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. |

**18.2.7.26 void I2C\_MasterTransferCreateHandle ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle*, i2c\_master\_transfer\_t *callback*, void \* *userData* )**

## Parameters

|                 |                                                                       |
|-----------------|-----------------------------------------------------------------------|
| <i>base</i>     | I2C base pointer.                                                     |
| <i>handle</i>   | pointer to i2c_master_handle_t structure to store the transfer state. |
| <i>callback</i> | pointer to user callback function.                                    |
| <i>userData</i> | user parameter passed to the callback function.                       |

**18.2.7.27 status\_t I2C\_MasterTransferNonBlocking ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *xfer* )**

## Note

Calling the API returns immediately after transfer initiates. The user needs to call I2C\_MasterGetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not *kStatus\_I2C\_Busy*, the transfer is finished.

Parameters

|               |                                                                           |
|---------------|---------------------------------------------------------------------------|
| <i>base</i>   | I2C base pointer.                                                         |
| <i>handle</i> | pointer to i2c_master_handle_t structure which stores the transfer state. |
| <i>xfer</i>   | pointer to i2c_master_transfer_t structure.                               |

Return values

|                            |                                           |
|----------------------------|-------------------------------------------|
| <i>kStatus_Success</i>     | Successfully start the data transmission. |
| <i>kStatus_I2C_Busy</i>    | Previous transmission still not finished. |
| <i>kStatus_I2C_Timeout</i> | Transfer error, wait signal timeout.      |

#### 18.2.7.28 status\_t I2C\_MasterTransferGetCount ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

|               |                                                                           |
|---------------|---------------------------------------------------------------------------|
| <i>base</i>   | I2C base pointer.                                                         |
| <i>handle</i> | pointer to i2c_master_handle_t structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.       |

Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_InvalidArgument</i> | <i>count</i> is Invalid.       |
| <i>kStatus_Success</i>         | Successfully return the count. |

#### 18.2.7.29 status\_t I2C\_MasterTransferAbort ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle* )

Note

This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | I2C base pointer.                                                        |
| <i>handle</i> | pointer to i2c_master_handle_t structure which stores the transfer state |

Return values

|                            |                                  |
|----------------------------|----------------------------------|
| <i>kStatus_I2C_Timeout</i> | Timeout during polling flag.     |
| <i>kStatus_Success</i>     | Successfully abort the transfer. |

#### 18.2.7.30 void I2C\_MasterTransferHandleIRQ ( *I2C\_Type* \* *base*, *void* \* *i2cHandle* )

Parameters

|                  |                                                  |
|------------------|--------------------------------------------------|
| <i>base</i>      | I2C base pointer.                                |
| <i>i2cHandle</i> | pointer to <i>i2c_master_handle_t</i> structure. |

#### 18.2.7.31 void I2C\_SlaveTransferCreateHandle ( *I2C\_Type* \* *base*, *i2c\_slave\_handle\_t* \* *handle*, *i2c\_slave\_transfer\_callback\_t* *callback*, *void* \* *userData* )

Parameters

|                 |                                                                             |
|-----------------|-----------------------------------------------------------------------------|
| <i>base</i>     | I2C base pointer.                                                           |
| <i>handle</i>   | pointer to <i>i2c_slave_handle_t</i> structure to store the transfer state. |
| <i>callback</i> | pointer to user callback function.                                          |
| <i>userData</i> | user parameter passed to the callback function.                             |

#### 18.2.7.32 *status\_t* I2C\_SlaveTransferNonBlocking ( *I2C\_Type* \* *base*, *i2c\_slave\_handle\_t* \* *handle*, *uint32\_t* *eventMask* )

Call this API after calling the [I2C\\_SlaveInit\(\)](#) and [I2C\\_SlaveTransferCreateHandle\(\)](#) to start processing transactions driven by an I2C master. The slave monitors the I2C bus and passes events to the callback that was passed into the call to [I2C\\_SlaveTransferCreateHandle\(\)](#). The callback is always invoked from the interrupt context.

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 k-I2C\_SlaveTransmitEvent and kLPI2C\_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, 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 i2c_slave_handle_t 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. |

### 18.2.7.33 void I2C\_SlaveTransferAbort ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle* )

Note

This API can be called at any time to stop slave for handling the bus events.

Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | I2C base pointer.                                                        |
| <i>handle</i> | pointer to i2c_slave_handle_t structure which stores the transfer state. |

### 18.2.7.34 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. |

18.2.7.35 void I2C\_SlaveTransferHandleIRQ ( I2C\_Type \* *base*, void \* *i2cHandle* )

## Parameters

|                  |                                                                         |
|------------------|-------------------------------------------------------------------------|
| <i>base</i>      | I2C base pointer.                                                       |
| <i>i2cHandle</i> | pointer to i2c_slave_handle_t structure which stores the transfer state |

## 18.3 I2C eDMA Driver

### 18.3.1 Overview

#### Data Structures

- struct [i2c\\_master\\_edma\\_handle\\_t](#)  
*I2C master eDMA transfer structure.* [More...](#)

#### TypeDefs

- [typedef void\(\\* i2c\\_master\\_edma\\_transfer\\_callback\\_t \)\(I2C\\_Type \\*base, i2c\\_master\\_edma\\_handle\\_t \\*handle, status\\_t status, void \\*userData\)](#)  
*I2C master eDMA transfer callback typedef.*

#### Driver version

- [#define FSL\\_I2C\\_EDMA\\_DRIVER\\_VERSION \(MAKE\\_VERSION\(2, 0, 9\)\)](#)  
*I2C EDMA driver version.*

## I2C Block eDMA Transfer Operation

- [void I2C\\_MasterCreateEDMAHandle \(I2C\\_Type \\*base, i2c\\_master\\_edma\\_handle\\_t \\*handle, i2c\\_master\\_edma\\_transfer\\_callback\\_t callback, void \\*userData, edma\\_handle\\_t \\*edmaHandle\)](#)  
*Initializes the I2C handle which is used in transactional functions.*
- [status\\_t I2C\\_MasterTransferEDMA \(I2C\\_Type \\*base, i2c\\_master\\_edma\\_handle\\_t \\*handle, i2c\\_master\\_transfer\\_t \\*xfer\)](#)  
*Performs a master eDMA non-blocking transfer on the I2C bus.*
- [status\\_t I2C\\_MasterTransferGetCountEDMA \(I2C\\_Type \\*base, i2c\\_master\\_edma\\_handle\\_t \\*handle, size\\_t \\*count\)](#)  
*Gets a master transfer status during the eDMA non-blocking transfer.*
- [void I2C\\_MasterTransferAbortEDMA \(I2C\\_Type \\*base, i2c\\_master\\_edma\\_handle\\_t \\*handle\)](#)  
*Aborts a master eDMA non-blocking transfer early.*

### 18.3.2 Data Structure Documentation

#### 18.3.2.1 struct \_i2c\_master\_edma\_handle

Retry times for waiting flag.

I2C master eDMA handle typedef.

## Data Fields

- `i2c_master_transfer_t transfer`  
*I2C master transfer structure.*
- `size_t transferSize`  
*Total bytes to be transferred.*
- `uint8_t nbytes`  
*eDMA minor byte transfer count initially configured.*
- `uint8_t state`  
*I2C master transfer status.*
- `edma_handle_t *dmaHandle`  
*The eDMA handler used.*
- `i2c_master_edma_transfer_callback_t completionCallback`  
*A callback function called after the eDMA transfer is finished.*
- `void *userData`  
*A callback parameter passed to the callback function.*

## Field Documentation

- (1) `i2c_master_transfer_t i2c_master_edma_handle_t::transfer`
- (2) `size_t i2c_master_edma_handle_t::transferSize`
- (3) `uint8_t i2c_master_edma_handle_t::nbytes`
- (4) `uint8_t i2c_master_edma_handle_t::state`
- (5) `edma_handle_t* i2c_master_edma_handle_t::dmaHandle`
- (6) `i2c_master_edma_transfer_callback_t i2c_master_edma_handle_t::completionCallback`
- (7) `void* i2c_master_edma_handle_t::userData`

### 18.3.3 Macro Definition Documentation

#### 18.3.3.1 `#define FSL_I2C_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 9))`

### 18.3.4 Typedef Documentation

#### 18.3.4.1 `typedef void(* i2c_master_edma_transfer_callback_t)(I2C_Type *base, i2c_master_edma_handle_t *handle, status_t status, void *userData)`

### 18.3.5 Function Documentation

#### 18.3.5.1 `void I2C_MasterCreateEDMAHandle ( I2C_Type * base, i2c_master_edma_handle_t * handle, i2c_master_edma_transfer_callback_t callback, void * userData, edma_handle_t * edmaHandle )`

Parameters

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>base</i>       | I2C peripheral base address.                         |
| <i>handle</i>     | A pointer to the i2c_master_edma_handle_t structure. |
| <i>callback</i>   | A pointer to the user callback function.             |
| <i>userData</i>   | A user parameter passed to the callback function.    |
| <i>edmaHandle</i> | eDMA handle pointer.                                 |

### 18.3.5.2 status\_t I2C\_MasterTransferEDMA ( I2C\_Type \* *base*, i2c\_master\_edma\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *xfer* )

Parameters

|               |                                                                                |
|---------------|--------------------------------------------------------------------------------|
| <i>base</i>   | I2C peripheral base address.                                                   |
| <i>handle</i> | A pointer to the i2c_master_edma_handle_t structure.                           |
| <i>xfer</i>   | A pointer to the transfer structure of <a href="#">i2c_master_transfer_t</a> . |

Return values

|                                     |                                                |
|-------------------------------------|------------------------------------------------|
| <i>kStatus_Success</i>              | Successfully completed the data transmission.  |
| <i>kStatus_I2C_Busy</i>             | A previous transmission is still not finished. |
| <i>kStatus_I2C_Timeout</i>          | Transfer error, waits for a signal timeout.    |
| <i>kStatus_I2C_Arbitration-Lost</i> | Transfer error, arbitration lost.              |
| <i>kStatus_I2C_Nak</i>              | Transfer error, receive NAK during transfer.   |

### 18.3.5.3 status\_t I2C\_MasterTransferGetCountEDMA ( I2C\_Type \* *base*, i2c\_master\_edma\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>base</i>   | I2C peripheral base address.                         |
| <i>handle</i> | A pointer to the i2c_master_edma_handle_t structure. |

|              |                                                                |
|--------------|----------------------------------------------------------------|
| <i>count</i> | A number of bytes transferred by the non-blocking transaction. |
|--------------|----------------------------------------------------------------|

#### 18.3.5.4 void I2C\_MasterTransferAbortEDMA ( I2C\_Type \* *base*, i2c\_master\_edma\_handle\_t \* *handle* )

Parameters

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>base</i>   | I2C peripheral base address.                         |
| <i>handle</i> | A pointer to the i2c_master_edma_handle_t structure. |

## 18.4 I2C FreeRTOS Driver

### 18.4.1 Overview

#### Driver version

- #define `FSL_I2C_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 9)`)  
*I2C FreeRTOS driver version 2.0.9.*

#### 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 the I2C transfer.*

### 18.4.2 Macro Definition Documentation

#### 18.4.2.1 #define `FSL_I2C_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 9)`)

### 18.4.3 Function Documentation

#### 18.4.3.1 `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.

Parameters

|                           |                                                                          |
|---------------------------|--------------------------------------------------------------------------|
| <code>handle</code>       | The RTOS I2C handle, the pointer to an allocated space for RTOS context. |
| <code>base</code>         | The pointer base address of the I2C instance to initialize.              |
| <code>masterConfig</code> | The configuration structure to set-up I2C in master mode.                |
| <code>srcClock_Hz</code>  | The frequency of an input clock of the I2C module.                       |

Returns

status of the operation.

#### 18.4.3.2 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. |
|---------------|----------------------|

#### **18.4.3.3 status\_t I2C\_RTOS\_Transfer ( i2c\_rtos\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *transfer* )**

This function performs the I2C transfer according to the data given in the transfer structure.

Parameters

|                 |                                                 |
|-----------------|-------------------------------------------------|
| <i>handle</i>   | The RTOS I2C handle.                            |
| <i>transfer</i> | A structure specifying the transfer parameters. |

Returns

status of the operation.

## 18.5 I2C CMSIS Driver

This section describes the programming interface of the I2C Cortex Microcontroller Software Interface Standard (CMSIS) driver. This driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method see <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

The I2C CMSIS driver includes transactional APIs.

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 accessing the hardware registers.

### 18.5.1 I2C CMSIS Driver

#### 18.5.1.1 Master Operation in interrupt transactional method

```
void I2C_MasterSignalEvent_t(uint32_t event)
{
 if (event == ARM_I2C_EVENT_TRANSFER_DONE)
 {
 g_MasterCompletionFlag = true;
 }
}
/*Init I2C0*/
Driver_I2C0.Initialize(I2C_MasterSignalEvent_t);

Driver_I2C0.PowerControl(ARM_POWER_FULL);

/*config transmit speed/
Driver_I2C0.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

/*start transmit*/
Driver_I2C0.MasterTransmit(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

/* Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

#### 18.5.1.2 Master Operation in DMA transactional method

```
void I2C_MasterSignalEvent_t(uint32_t event)
{
 /* Transfer done */
 if (event == ARM_I2C_EVENT_TRANSFER_DONE)
 {
 g_MasterCompletionFlag = true;
 }
}

/* Init DMAMUX and DMA/EDMA. */
DMAMUX_Init(EXAMPLE_I2C_DMAMUX_BASEADDR)
```

```

#if defined(FSL_FEATURE_SOC_DMA_COUNT) && FSL_FEATURE_SOC_DMA_COUNT > 0U
 DMA_Init(EXAMPLE_I2C_DMA_BASEADDR);
#endif /* FSL_FEATURE_SOC_DMA_COUNT */

#if defined(FSL_FEATURE_SOC_EDMA_COUNT) && FSL_FEATURE_SOC_EDMA_COUNT > 0U
 edma_config_t edmaConfig;

 EDMA_GetDefaultConfig(&edmaConfig);
 EDMA_Init(EXAMPLE_I2C_DMA_BASEADDR, &edmaConfig);
#endif /* FSL_FEATURE_SOC_EDMA_COUNT */

/*Init I2C0*/
Driver_I2C0.Initialize(I2C_MasterSignalEvent_t);

Driver_I2C0.PowerControl(ARM_POWER_FULL);

/*config transmit speed*/
Driver_I2C0.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

/*start transfer*/
Driver_I2C0.MasterReceive(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

/* Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;

```

### 18.5.1.3 Slave Operation in interrupt transactional method

```

void I2C_SlaveSignalEvent_t(uint32_t event)
{
 /* Transfer done */
 if (event == ARM_I2C_EVENT_TRANSFER_DONE)
 {
 g_SlaveCompletionFlag = true;
 }
}

/*Init I2C1*/
Driver_I2C1.Initialize(I2C_SlaveSignalEvent_t);

Driver_I2C1.PowerControl(ARM_POWER_FULL);

/*config slave addr*/
Driver_I2C1.Control(ARM_I2C_OWN_ADDRESS, I2C_MASTER_SLAVE_ADDR);

/*start transfer*/
Driver_I2C1.SlaveReceive(g_slave_buff, I2C_DATA_LENGTH);

/* Wait for transfer completed. */
while (!g_SlaveCompletionFlag)
{
}
g_SlaveCompletionFlag = false;

```

# Chapter 19

## LLWU: Low-Leakage Wakeup Unit Driver

### 19.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low-Leakage Wakeup Unit (LLWU) module of MCUXpresso SDK devices. The LLWU module allows the user to select external pin sources and internal modules as a wake-up source from low-leakage power modes.

### 19.2 External wakeup pins configurations

Configures the external wakeup pins' working modes, gets, and clears the wake pin flags. External wakeup pins are accessed by the `pinIndex`, which is started from 1. Numbers of the external pins depend on the SoC configuration.

### 19.3 Internal wakeup modules configurations

Enables/disables the internal wakeup modules and gets the module flags. Internal modules are accessed by `moduleIndex`, which is started from 1. Numbers of external pins depend the on SoC configuration.

### 19.4 Digital pin filter for external wakeup pin configurations

Configures the digital pin filter of the external wakeup pins' working modes, gets, and clears the pin filter flags. Digital pin filters are accessed by the `filterIndex`, which is started from 1. Numbers of external pins depend on the SoC configuration.

## Data Structures

- struct `llwu_external_pin_filter_mode_t`  
*An external input pin filter control structure. [More...](#)*

## Enumerations

- enum `llwu_external_pin_mode_t` {  
  `kLLWU_ExternalPinDisable` = 0U,  
  `kLLWU_ExternalPinRisingEdge` = 1U,  
  `kLLWU_ExternalPinFallingEdge` = 2U,  
  `kLLWU_ExternalPinAnyEdge` = 3U }  
*External input pin control modes.*
- enum `llwu_pin_filter_mode_t` {  
  `kLLWU_PinFilterDisable` = 0U,  
  `kLLWU_PinFilterRisingEdge` = 1U,  
  `kLLWU_PinFilterFallingEdge` = 2U,  
  `kLLWU_PinFilterAnyEdge` = 3U }  
*Digital filter control modes.*

## Driver version

- #define `FSL_LLWU_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 5)`)  
*LLWU driver version.*

## Low-Leakage Wakeup Unit Control APIs

- void `LLWU_SetExternalWakePinMode` (`LLWU_Type *base`, `uint32_t pinIndex`, `llwu_external_pin_mode_t pinMode`)  
*Sets the external input pin source mode.*
- bool `LLWU_GetExternalWakePinFlag` (`LLWU_Type *base`, `uint32_t pinIndex`)  
*Gets the external wakeup source flag.*
- void `LLWU_ClearExternalWakePinFlag` (`LLWU_Type *base`, `uint32_t pinIndex`)  
*Clears the external wakeup source flag.*
- static void `LLWU_EnableInternalModuleInterruptWakeup` (`LLWU_Type *base`, `uint32_t moduleIndex`, `bool enable`)  
*Enables/disables the internal module source.*
- static bool `LLWU_GetInternalWakeModuleFlag` (`LLWU_Type *base`, `uint32_t moduleIndex`)  
*Gets the external wakeup source flag.*
- void `LLWU_SetPinFilterMode` (`LLWU_Type *base`, `uint32_t filterIndex`, `llwu_external_pin_filter_mode_t filterMode`)  
*Sets the pin filter configuration.*
- bool `LLWU_GetPinFilterFlag` (`LLWU_Type *base`, `uint32_t filterIndex`)  
*Gets the pin filter configuration.*
- void `LLWU_ClearPinFilterFlag` (`LLWU_Type *base`, `uint32_t filterIndex`)  
*Clears the pin filter configuration.*
- #define `INTERNAL_WAKEUP_MODULE_FLAG_REG` F3

## 19.5 Data Structure Documentation

### 19.5.1 struct `llwu_external_pin_filter_mode_t`

#### Data Fields

- `uint32_t pinIndex`  
*A pin number.*
- `llwu_pin_filter_mode_t filterMode`  
*Filter mode.*

## 19.6 Macro Definition Documentation

### 19.6.1 #define `FSL_LLWU_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 5)`)

## 19.7 Enumeration Type Documentation

### 19.7.1 enum `llwu_external_pin_mode_t`

Enumerator

***kLLWU\_ExternalPinDisable*** Pin disabled as a wakeup input.

*kLLWU\_ExternalPinRisingEdge* Pin enabled with the rising edge detection.

*kLLWU\_ExternalPinFallingEdge* Pin enabled with the falling edge detection.

*kLLWU\_ExternalPinAnyEdge* Pin enabled with any change detection.

## 19.7.2 enum llwu\_pin\_filter\_mode\_t

Enumerator

*kLLWU\_PinFilterDisable* Filter disabled.

*kLLWU\_PinFilterRisingEdge* Filter positive edge detection.

*kLLWU\_PinFilterFallingEdge* Filter negative edge detection.

*kLLWU\_PinFilterAnyEdge* Filter any edge detection.

## 19.8 Function Documentation

### 19.8.1 void LLWU\_SetExternalWakeupsPinMode ( *LLWU\_Type \* base*, *uint32\_t pinIndex*, *llwu\_external\_pin\_mode\_t pinMode* )

This function sets the external input pin source mode that is used as a wake up source.

Parameters

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <i>base</i>     | LLWU peripheral base address.                                           |
| <i>pinIndex</i> | A pin index to be enabled as an external wakeup source starting from 1. |
| <i>pinMode</i>  | A pin configuration mode defined in the llwu_external_pin_modes_t.      |

### 19.8.2 bool LLWU\_GetExternalWakeupsPinFlag ( *LLWU\_Type \* base*, *uint32\_t pinIndex* )

This function checks the external pin flag to detect whether the MCU is woken up by the specific pin.

Parameters

|                 |                                   |
|-----------------|-----------------------------------|
| <i>base</i>     | LLWU peripheral base address.     |
| <i>pinIndex</i> | A pin index, which starts from 1. |

Returns

True if the specific pin is a wakeup source.

### 19.8.3 void LLWU\_ClearExternalWakeupPinFlag ( **LLWU\_Type** \* *base*, **uint32\_t** *pinIndex* )

This function clears the external wakeup source flag for a specific pin.

Parameters

|                 |                                   |
|-----------------|-----------------------------------|
| <i>base</i>     | LLWU peripheral base address.     |
| <i>pinIndex</i> | A pin index, which starts from 1. |

#### 19.8.4 static void LLWU\_EnableInternalModuleInterruptWakup ( LLWU\_Type \* *base*, uint32\_t *moduleIndex*, bool *enable* ) [inline], [static]

This function enables/disables the internal module source mode that is used as a wake up source.

Parameters

|                    |                                                                            |
|--------------------|----------------------------------------------------------------------------|
| <i>base</i>        | LLWU peripheral base address.                                              |
| <i>moduleIndex</i> | A module index to be enabled as an internal wakeup source starting from 1. |
| <i>enable</i>      | An enable or a disable setting                                             |

#### 19.8.5 static bool LLWU\_GetInternalWakeupModuleFlag ( LLWU\_Type \* *base*, uint32\_t *moduleIndex* ) [inline], [static]

This function checks the external pin flag to detect whether the system is woken up by the specific pin.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>base</i>        | LLWU peripheral base address.        |
| <i>moduleIndex</i> | A module index, which starts from 1. |

Returns

True if the specific pin is a wake up source.

#### 19.8.6 void LLWU\_SetPinFilterMode ( LLWU\_Type \* *base*, uint32\_t *filterIndex*, llwu\_external\_pin\_filter\_mode\_t *filterMode* )

This function sets the pin filter configuration.

Parameters

|                    |                                                                                |
|--------------------|--------------------------------------------------------------------------------|
| <i>base</i>        | LLWU peripheral base address.                                                  |
| <i>filterIndex</i> | A pin filter index used to enable/disable the digital filter, starting from 1. |
| <i>filterMode</i>  | A filter mode configuration                                                    |

### 19.8.7 **bool LLWU\_GetPinFilterFlag ( LLWU\_Type \* *base*, uint32\_t *filterIndex* )**

This function gets the pin filter flag.

Parameters

|                    |                                          |
|--------------------|------------------------------------------|
| <i>base</i>        | LLWU peripheral base address.            |
| <i>filterIndex</i> | A pin filter index, which starts from 1. |

Returns

True if the flag is a source of the existing low-leakage power mode.

### 19.8.8 **void LLWU\_ClearPinFilterFlag ( LLWU\_Type \* *base*, uint32\_t *filterIndex* )**

This function clears the pin filter flag.

Parameters

|                    |                                                        |
|--------------------|--------------------------------------------------------|
| <i>base</i>        | LLWU peripheral base address.                          |
| <i>filterIndex</i> | A pin filter index to clear the flag, starting from 1. |

# Chapter 20

## LPTMR: Low-Power Timer

### 20.1 Overview

The MCUXpresso SDK provides a driver for the Low-Power Timer (LPTMR) of MCUXpresso SDK devices.

### 20.2 Function groups

The LPTMR driver supports operating the module as a time counter or as a pulse counter.

#### 20.2.1 Initialization and deinitialization

The function [LPTMR\\_Init\(\)](#) initializes the LPTMR with specified configurations. The function [LPTMR\\_GetDefaultConfig\(\)](#) gets the default configurations. The initialization function configures the LPTMR for a timer or a pulse counter mode mode. It also sets up the LPTMR's free running mode operation and a clock source.

The function [LPTMR\\_DeInit\(\)](#) disables the LPTMR module and gates the module clock.

#### 20.2.2 Timer period Operations

The function [LPTMR\\_SetTimerPeriod\(\)](#) sets the timer period in units of count. Timers counts from 0 to the count value set here.

The function [LPTMR\\_GetCurrentTimerCount\(\)](#) reads the current timer counting value. This function returns the real-time timer counting value ranging from 0 to a timer period.

The timer period operation function takes the count value in ticks. Call the utility macros provided in the `fsl_common.h` file to convert to microseconds or milliseconds.

#### 20.2.3 Start and Stop timer operations

The function [LPTMR\\_StartTimer\(\)](#) starts the timer counting. After calling this function, the timer counts up to the counter value set earlier by using the [LPTMR\\_SetPeriod\(\)](#) function. Each time the timer reaches the count value and increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

The function [LPTMR\\_StopTimer\(\)](#) stops the timer counting and resets the timer's counter register.

## 20.2.4 Status

Provides functions to get and clear the LPTMR status.

## 20.2.5 Interrupt

Provides functions to enable/disable LPTMR interrupts and get the currently enabled interrupts.

## 20.3 Typical use case

### 20.3.1 LPTMR tick example

Updates the LPTMR period and toggles an LED periodically. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/lptmr

## Data Structures

- struct [lptmr\\_config\\_t](#)  
*LPTMR config structure.* [More...](#)

## Enumerations

- enum [lptmr\\_pin\\_select\\_t](#) {
   
   kLPTMR\_PinSelectInput\_0 = 0x0U,
   
   kLPTMR\_PinSelectInput\_1 = 0x1U,
   
   kLPTMR\_PinSelectInput\_2 = 0x2U,
   
   kLPTMR\_PinSelectInput\_3 = 0x3U }
   
*LPTMR pin selection used in pulse counter mode.*
- enum [lptmr\\_pin\\_polarity\\_t](#) {
   
   kLPTMR\_PinPolarityActiveHigh = 0x0U,
   
   kLPTMR\_PinPolarityActiveLow = 0x1U }
   
*LPTMR pin polarity used in pulse counter mode.*
- enum [lptmr\\_timer\\_mode\\_t](#) {
   
   kLPTMR\_TimerModeTimeCounter = 0x0U,
   
   kLPTMR\_TimerModePulseCounter = 0x1U }
   
*LPTMR timer mode selection.*
- enum [lptmr\\_prescaler\\_glitch\\_value\\_t](#) {

```

kLPTMR_Prescale_Glitch_0 = 0x0U,
kLPTMR_Prescale_Glitch_1 = 0x1U,
kLPTMR_Prescale_Glitch_2 = 0x2U,
kLPTMR_Prescale_Glitch_3 = 0x3U,
kLPTMR_Prescale_Glitch_4 = 0x4U,
kLPTMR_Prescale_Glitch_5 = 0x5U,
kLPTMR_Prescale_Glitch_6 = 0x6U,
kLPTMR_Prescale_Glitch_7 = 0x7U,
kLPTMR_Prescale_Glitch_8 = 0x8U,
kLPTMR_Prescale_Glitch_9 = 0x9U,
kLPTMR_Prescale_Glitch_10 = 0xAU,
kLPTMR_Prescale_Glitch_11 = 0xBU,
kLPTMR_Prescale_Glitch_12 = 0xCU,
kLPTMR_Prescale_Glitch_13 = 0xDU,
kLPTMR_Prescale_Glitch_14 = 0xEU,
kLPTMR_Prescale_Glitch_15 = 0xFU }

```

*LPTMR prescaler/glitch filter values.*

- enum `lptmr_prescaler_clock_select_t` {
   
kLPTMR\_PrescalerClock\_0 = 0x0U,
 kLPTMR\_PrescalerClock\_1 = 0x1U,
 kLPTMR\_PrescalerClock\_2 = 0x2U,
 kLPTMR\_PrescalerClock\_3 = 0x3U }

*LPTMR prescaler/glitch filter clock select.*

- enum `lptmr_interrupt_enable_t` { `kLPTMR_TimerInterruptEnable` = LPTMR\_CSR\_TIE\_MASK }
- List of the LPTMR interrupts.*
- enum `lptmr_status_flags_t` { `kLPTMR_TimerCompareFlag` = LPTMR\_CSR\_TCF\_MASK }
- List of the LPTMR status flags.*

## Driver version

- #define `FSL_LPTMR_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 1)`)
   
*Version 2.1.1.*

## Initialization and deinitialization

- void `LPTMR_Init` (LPTMR\_Type \*base, const `lptmr_config_t` \*config)
   
*Ungates the LPTMR clock and configures the peripheral for a basic operation.*
- void `LPTMR_Deinit` (LPTMR\_Type \*base)
   
*Gates the LPTMR clock.*
- void `LPTMR_GetDefaultConfig` (`lptmr_config_t` \*config)
   
*Fills in the LPTMR configuration structure with default settings.*

## Interrupt Interface

- static void `LPTMR_EnableInterrupts` (LPTMR\_Type \*base, uint32\_t mask)
   
*Enables the selected LPTMR interrupts.*
- static void `LPTMR_DisableInterrupts` (LPTMR\_Type \*base, uint32\_t mask)
   
*Disables the selected LPTMR interrupts.*

- static uint32\_t [LPTMR\\_GetEnabledInterrupts](#) (LPTMR\_Type \*base)  
*Gets the enabled LPTMR interrupts.*

## Status Interface

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

## Read and write the timer period

- static void [LPTMR\\_SetTimerPeriod](#) (LPTMR\_Type \*base, uint32\_t ticks)  
*Sets the timer period in units of count.*
- static uint32\_t [LPTMR\\_GetCurrentTimerCount](#) (LPTMR\_Type \*base)  
*Reads the current timer counting value.*

## Timer Start and Stop

- static void [LPTMR\\_StartTimer](#) (LPTMR\_Type \*base)  
*Starts the timer.*
- static void [LPTMR\\_StopTimer](#) (LPTMR\_Type \*base)  
*Stops the timer.*

## 20.4 Data Structure Documentation

### 20.4.1 struct lptmr\_config\_t

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

The configuration struct can be made constant so it resides in flash.

## Data Fields

- [lptmr\\_timer\\_mode\\_t timerMode](#)  
*Time counter mode or pulse counter mode.*
- [lptmr\\_pin\\_select\\_t pinSelect](#)  
*LPTMR pulse input pin select; used only in pulse counter mode.*
- [lptmr\\_pin\\_polarity\\_t pinPolarity](#)  
*LPTMR pulse input pin polarity; used only in pulse counter mode.*
- bool [enableFreeRunning](#)  
*True: enable free running, counter is reset on overflow False: counter is reset when the compare flag is set.*
- bool [bypassPrescaler](#)  
*True: bypass prescaler; false: use clock from prescaler.*
- [lptmr\\_prescaler\\_clock\\_select\\_t prescalerClockSource](#)

- [lptmr\\_prescaler\\_glitch\\_value\\_t](#) value  
*Prescaler or glitch filter value.*

## 20.5 Enumeration Type Documentation

### 20.5.1 enum lptmr\_pin\_select\_t

Enumerator

- kLPTMR\_PinSelectInput\_0*** Pulse counter input 0 is selected.
- kLPTMR\_PinSelectInput\_1*** Pulse counter input 1 is selected.
- kLPTMR\_PinSelectInput\_2*** Pulse counter input 2 is selected.
- kLPTMR\_PinSelectInput\_3*** Pulse counter input 3 is selected.

### 20.5.2 enum lptmr\_pin\_polarity\_t

Enumerator

- kLPTMR\_PinPolarityActiveHigh*** Pulse Counter input source is active-high.
- kLPTMR\_PinPolarityActiveLow*** Pulse Counter input source is active-low.

### 20.5.3 enum lptmr\_timer\_mode\_t

Enumerator

- kLPTMR\_TimerModeTimeCounter*** Time Counter mode.
- kLPTMR\_TimerModePulseCounter*** Pulse Counter mode.

### 20.5.4 enum lptmr\_prescaler\_glitch\_value\_t

Enumerator

- kLPTMR\_Prescale\_Glitch\_0*** Prescaler divide 2, glitch filter does not support this setting.
- kLPTMR\_Prescale\_Glitch\_1*** Prescaler divide 4, glitch filter 2.
- kLPTMR\_Prescale\_Glitch\_2*** Prescaler divide 8, glitch filter 4.
- kLPTMR\_Prescale\_Glitch\_3*** Prescaler divide 16, glitch filter 8.
- kLPTMR\_Prescale\_Glitch\_4*** Prescaler divide 32, glitch filter 16.
- kLPTMR\_Prescale\_Glitch\_5*** Prescaler divide 64, glitch filter 32.
- kLPTMR\_Prescale\_Glitch\_6*** Prescaler divide 128, glitch filter 64.
- kLPTMR\_Prescale\_Glitch\_7*** Prescaler divide 256, glitch filter 128.
- kLPTMR\_Prescale\_Glitch\_8*** Prescaler divide 512, glitch filter 256.

- kLPTMR\_Prescale\_Glitch\_9* Prescaler divide 1024, glitch filter 512.
- kLPTMR\_Prescale\_Glitch\_10* Prescaler divide 2048 glitch filter 1024.
- kLPTMR\_Prescale\_Glitch\_11* Prescaler divide 4096, glitch filter 2048.
- kLPTMR\_Prescale\_Glitch\_12* Prescaler divide 8192, glitch filter 4096.
- kLPTMR\_Prescale\_Glitch\_13* Prescaler divide 16384, glitch filter 8192.
- kLPTMR\_Prescale\_Glitch\_14* Prescaler divide 32768, glitch filter 16384.
- kLPTMR\_Prescale\_Glitch\_15* Prescaler divide 65536, glitch filter 32768.

## 20.5.5 enum lptmr\_prescaler\_clock\_select\_t

Note

Clock connections are SoC-specific

Enumerator

- kLPTMR\_PrescalerClock\_0* Prescaler/glitch filter clock 0 selected.
- kLPTMR\_PrescalerClock\_1* Prescaler/glitch filter clock 1 selected.
- kLPTMR\_PrescalerClock\_2* Prescaler/glitch filter clock 2 selected.
- kLPTMR\_PrescalerClock\_3* Prescaler/glitch filter clock 3 selected.

## 20.5.6 enum lptmr\_interrupt\_enable\_t

Enumerator

*kLPTMR\_TimerInterruptEnable* Timer interrupt enable.

## 20.5.7 enum lptmr\_status\_flags\_t

Enumerator

*kLPTMR\_TimerCompareFlag* Timer compare flag.

## 20.6 Function Documentation

### 20.6.1 void LPTMR\_Init ( LPTMR\_Type \* *base*, const lptmr\_config\_t \* *config* )

Note

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

Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>base</i>   | LPTMR peripheral base address                   |
| <i>config</i> | A pointer to the LPTMR configuration structure. |

## 20.6.2 void LPTMR\_Deinit ( LPTMR\_Type \* *base* )

Parameters

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

## 20.6.3 void LPTMR\_GetDefaultConfig ( lptmr\_config\_t \* *config* )

The default values are as follows.

```
* config->timerMode = kLPTMR_TimerModeTimeCounter;
* config->pinSelect = kLPTMR_PinSelectInput_0;
* config->pinPolarity = kLPTMR_PinPolarityActiveHigh;
* config->enableFreeRunning = false;
* config->bypassPrescaler = true;
* config->prescalerClockSource = kLPTMR_PrescalerClock_1;
* config->value = kLPTMR_Prescale_Glitch_0;
*
```

Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>config</i> | A pointer to the LPTMR configuration structure. |
|---------------|-------------------------------------------------|

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

Parameters

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

## 20.6.5 static void LPTMR\_DisableInterrupts ( LPTMR\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | LPTMR peripheral base address                                                                                            |
| <i>mask</i> | The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">lptmr_interrupt_enable_t</a> . |

## 20.6.6 static uint32\_t LPTMR\_GetEnabledInterrupts ( LPTMR\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

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

## 20.6.7 static uint32\_t LPTMR\_GetStatusFlags ( LPTMR\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

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

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

Parameters

---

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

### 20.6.9 static void LPTMR\_SetTimerPeriod ( LPTMR\_Type \* *base*, uint32\_t *ticks* ) [inline], [static]

Timers counts from 0 until it equals the count value set here. The count value is written to the CMR register.

Note

1. The TCF flag is set with the CNR equals the count provided here and then increments.
2. Call the utility macros provided in the `fsl_common.h` to convert to ticks.

Parameters

|              |                                                                            |
|--------------|----------------------------------------------------------------------------|
| <i>base</i>  | LPTMR peripheral base address                                              |
| <i>ticks</i> | A timer period in units of ticks, which should be equal or greater than 1. |

### 20.6.10 static uint32\_t LPTMR\_GetCurrentTimerCount ( LPTMR\_Type \* *base* ) [inline], [static]

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note

Call the utility macros provided in the `fsl_common.h` to convert ticks to usec or msec.

Parameters

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

Returns

The current counter value in ticks

**20.6.11 static void LPTMR\_StartTimer ( LPTMR\_Type \* *base* ) [inline],  
[static]**

After calling this function, the timer counts up to the CMR register value. Each time the timer reaches the CMR value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

Parameters

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

**20.6.12 static void LPTMR\_StopTimer ( LPTMR\_Type \* *base* ) [inline],  
[static]**

This function stops the timer and resets the timer's counter register.

Parameters

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

## Chapter 21

# LPUART: Low Power Universal Asynchronous Receiver-/Transmitter Driver

### 21.1 Overview

#### Modules

- [LPUART CMSIS Driver](#)
- [LPUART Driver](#)
- [LPUART FreeRTOS Driver](#)
- [LPUART eDMA Driver](#)

## 21.2 LPUART Driver

### 21.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power UART (LPUART) module of MCUXpresso SDK devices.

### 21.2.2 Typical use case

#### 21.2.2.1 LPUART Operation

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/lpuart

## Data Structures

- struct [lpuart\\_config\\_t](#)  
*LPUART configuration structure. [More...](#)*
- struct [lpuart\\_transfer\\_t](#)  
*LPUART transfer structure. [More...](#)*
- struct [lpuart\\_handle\\_t](#)  
*LPUART handle structure. [More...](#)*

## Macros

- #define [UART\\_RETRY\\_TIMES](#) 0U /\* Defining to zero means to keep waiting for the flag until it is assert/deassert. \*/  
*Retry times for waiting flag.*

## Typedefs

- typedef void(\* [lpuart\\_transfer\\_callback\\_t](#))(LPUART\_Type \*base, lpuart\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*LPUART transfer callback function.*

## Enumerations

- enum {
   
kStatus\_LPUART\_TxBusy = MAKE\_STATUS(kStatusGroup\_LPUART, 0),
   
kStatus\_LPUART\_RxBusy = MAKE\_STATUS(kStatusGroup\_LPUART, 1),
   
kStatus\_LPUART\_TxIdle = MAKE\_STATUS(kStatusGroup\_LPUART, 2),
   
kStatus\_LPUART\_RxIdle = MAKE\_STATUS(kStatusGroup\_LPUART, 3),
   
kStatus\_LPUART\_TxWatermarkTooLarge = MAKE\_STATUS(kStatusGroup\_LPUART, 4),
   
kStatus\_LPUART\_RxWatermarkTooLarge = MAKE\_STATUS(kStatusGroup\_LPUART, 5),
   
kStatus\_LPUART\_FlagCannotClearManually = MAKE\_STATUS(kStatusGroup\_LPUART, 6),
   
kStatus\_LPUART\_Error = MAKE\_STATUS(kStatusGroup\_LPUART, 7),
   
kStatus\_LPUART\_RxRingBufferOverrun,
   
kStatus\_LPUART\_RxHardwareOverrun = MAKE\_STATUS(kStatusGroup\_LPUART, 9),
   
kStatus\_LPUART\_NoiseError = MAKE\_STATUS(kStatusGroup\_LPUART, 10),
   
kStatus\_LPUART\_FramingError = MAKE\_STATUS(kStatusGroup\_LPUART, 11),
   
kStatus\_LPUART\_ParityError = MAKE\_STATUS(kStatusGroup\_LPUART, 12),
   
kStatus\_LPUART\_BaudrateNotSupport,
   
kStatus\_LPUART\_IdleLineDetected = MAKE\_STATUS(kStatusGroup\_LPUART, 14),
   
kStatus\_LPUART\_Timeout = MAKE\_STATUS(kStatusGroup\_LPUART, 15) }

*Error codes for the LPUART driver.*

- enum `lpuart_parity_mode_t` {
   
kLPUART\_ParityDisabled = 0x0U,
   
kLPUART\_ParityEven = 0x2U,
   
kLPUART\_ParityOdd = 0x3U }
- LPUART parity mode.*
- enum `lpuart_data_bits_t` { kLPUART\_EightDataBits = 0x0U }
- LPUART data bits count.*
- enum `lpuart_stop_bit_count_t` {
   
kLPUART\_OneStopBit = 0U,
   
kLPUART\_TwoStopBit = 1U }
- LPUART stop bit count.*
- enum `lpuart_transmit_cts_source_t` {
   
kLPUART\_CtsSourcePin = 0U,
   
kLPUART\_CtsSourceMatchResult = 1U }
- LPUART transmit CTS source.*
- enum `lpuart_transmit_cts_config_t` {
   
kLPUART\_CtsSampleAtStart = 0U,
   
kLPUART\_CtsSampleAtIdle = 1U }
- LPUART transmit CTS configure.*
- enum `lpuart_idle_type_select_t` {
   
kLPUART\_IdleTypeStartBit = 0U,
   
kLPUART\_IdleTypeStopBit = 1U }
- LPUART idle flag type defines when the receiver starts counting.*
- enum `lpuart_idle_config_t` {

```
kLPUART_IdleCharacter1 = 0U,
kLPUART_IdleCharacter2 = 1U,
kLPUART_IdleCharacter4 = 2U,
kLPUART_IdleCharacter8 = 3U,
kLPUART_IdleCharacter16 = 4U,
kLPUART_IdleCharacter32 = 5U,
kLPUART_IdleCharacter64 = 6U,
kLPUART_IdleCharacter128 = 7U }
```

*LPUART idle detected configuration.*

- enum `_lpuart_interrupt_enable` {
 

```
kLPUART_LinBreakInterruptEnable = (LPUART_BAUD_LBKDIIE_MASK >> 8U),
kLPUART_RxActiveEdgeInterruptEnable = (LPUART_BAUD_RXEDGIE_MASK >> 8U),
kLPUART_TxDataRegEmptyInterruptEnable = (LPUART_CTRL_TIE_MASK),
kLPUART_TransmissionCompleteInterruptEnable = (LPUART_CTRL_TCIE_MASK),
kLPUART_RxDataRegFullInterruptEnable = (LPUART_CTRL_RIE_MASK),
kLPUART_IdleLineInterruptEnable = (LPUART_CTRL_ILIE_MASK),
kLPUART_RxOverrunInterruptEnable = (LPUART_CTRL_ORIE_MASK),
kLPUART_NoiseErrorInterruptEnable = (LPUART_CTRL_NEIE_MASK),
kLPUART_FramingErrorInterruptEnable = (LPUART_CTRL_FEIE_MASK),
kLPUART_ParityErrorInterruptEnable = (LPUART_CTRL_PEIE_MASK),
kLPUART_Match1InterruptEnable = (LPUART_CTRL_MA1IE_MASK),
kLPUART_Match2InterruptEnable = (LPUART_CTRL_MA2IE_MASK) }
```

*LPUART interrupt configuration structure, default settings all disabled.*

- enum `_lpuart_flags` {
 

```
kLPUART_TxDataRegEmptyFlag,
kLPUART_TransmissionCompleteFlag,
kLPUART_RxDataRegFullFlag = (LPUART_STAT_RDRF_MASK),
kLPUART_IdleLineFlag = (LPUART_STAT_IDLE_MASK),
kLPUART_RxOverrunFlag = (LPUART_STAT_OR_MASK),
kLPUART_NoiseErrorFlag = (LPUART_STAT_NF_MASK),
kLPUART_FramingErrorFlag,
kLPUART_ParityErrorFlag = (LPUART_STAT_PF_MASK),
kLPUART_LinBreakFlag = (LPUART_STAT_LBKDIF_MASK),
kLPUART_RxActiveEdgeFlag = (LPUART_STAT_RXEDGIF_MASK),
kLPUART_RxActiveFlag,
kLPUART_DataMatch1Flag,
kLPUART_DataMatch2Flag }
```

*LPUART status flags.*

## Driver version

- #define `FSL_LPUART_DRIVER_VERSION` (`MAKE_VERSION(2, 5, 3)`)

*LPUART driver version.*

## Initialization and deinitialization

- **status\_t LPUART\_Init** (LPUART\_Type \*base, const lpuart\_config\_t \*config, uint32\_t srcClock\_Hz)

*Initializes an LPUART instance with the user configuration structure and the peripheral clock.*

- **void LPUART\_Deinit** (LPUART\_Type \*base)

*Deinitializes a LPUART instance.*

- **void LPUART\_GetDefaultConfig** (lpuart\_config\_t \*config)

*Gets the default configuration structure.*

## Module configuration

- **status\_t LPUART\_SetBaudRate** (LPUART\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)

*Sets the LPUART instance baudrate.*

- **void LPUART\_Enable9bitMode** (LPUART\_Type \*base, bool enable)

*Enable 9-bit data mode for LPUART.*

- **static void LPUART\_SetMatchAddress** (LPUART\_Type \*base, uint16\_t address1, uint16\_t address2)

*Set the LPUART address.*

- **static void LPUART\_EnableMatchAddress** (LPUART\_Type \*base, bool match1, bool match2)

*Enable the LPUART match address feature.*

## Status

- **uint32\_t LPUART\_GetStatusFlags** (LPUART\_Type \*base)

*Gets LPUART status flags.*

- **status\_t LPUART\_ClearStatusFlags** (LPUART\_Type \*base, uint32\_t mask)

*Clears status flags with a provided mask.*

## Interrupts

- **void LPUART\_EnableInterrupts** (LPUART\_Type \*base, uint32\_t mask)

*Enables LPUART interrupts according to a provided mask.*

- **void LPUART\_DisableInterrupts** (LPUART\_Type \*base, uint32\_t mask)

*Disables LPUART interrupts according to a provided mask.*

- **uint32\_t LPUART\_GetEnabledInterrupts** (LPUART\_Type \*base)

*Gets enabled LPUART interrupts.*

## DMA Configuration

- **static uint32\_t LPUART\_GetDataRegisterAddress** (LPUART\_Type \*base)

*Gets the LPUART data register address.*

- **static void LPUART\_EnableTxDMA** (LPUART\_Type \*base, bool enable)

*Enables or disables the LPUART transmitter DMA request.*

- static void [LPUART\\_EnableRxDMA](#) (LPUART\_Type \*base, bool enable)  
*Enables or disables the LPUART receiver DMA.*

## Bus Operations

- uint32\_t [LPUART\\_GetInstance](#) (LPUART\_Type \*base)  
*Get the LPUART instance from peripheral base address.*
- static void [LPUART\\_EnableTx](#) (LPUART\_Type \*base, bool enable)  
*Enables or disables the LPUART transmitter.*
- static void [LPUART\\_EnableRx](#) (LPUART\_Type \*base, bool enable)  
*Enables or disables the LPUART receiver.*
- static void [LPUART\\_WriteByte](#) (LPUART\_Type \*base, uint8\_t data)  
*Writes to the transmitter register.*
- static uint8\_t [LPUART\\_ReadByte](#) (LPUART\_Type \*base)  
*Reads the receiver register.*
- void [LPUART\\_SendAddress](#) (LPUART\_Type \*base, uint8\_t address)  
*Transmit an address frame in 9-bit data mode.*
- status\_t [LPUART\\_WriteBlocking](#) (LPUART\_Type \*base, const uint8\_t \*data, size\_t length)  
*Writes to the transmitter register using a blocking method.*
- status\_t [LPUART\\_ReadBlocking](#) (LPUART\_Type \*base, uint8\_t \*data, size\_t length)  
*Reads the receiver data register using a blocking method.*

## Transactional

- void [LPUART\\_TransferCreateHandle](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle, [lpuart\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the LPUART handle.*
- status\_t [LPUART\\_TransferSendNonBlocking](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle, [lpuart\\_transfer\\_t](#) \*xfer)  
*Transmits a buffer of data using the interrupt method.*
- void [LPUART\\_TransferStartRingBuffer](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle, uint8\_t \*ringBuffer, size\_t ringBufferSize)  
*Sets up the RX ring buffer.*
- void [LPUART\\_TransferStopRingBuffer](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle)  
*Aborts the background transfer and uninstalls the ring buffer.*
- size\_t [LPUART\\_TransferGetRxRingBufferLength](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle)  
*Get the length of received data in RX ring buffer.*
- void [LPUART\\_TransferAbortSend](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle)  
*Aborts the interrupt-driven data transmit.*
- status\_t [LPUART\\_TransferGetSendCount](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of bytes that have been sent out to bus.*
- status\_t [LPUART\\_TransferReceiveNonBlocking](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle, [lpuart\\_transfer\\_t](#) \*xfer, size\_t \*receivedBytes)  
*Receives a buffer of data using the interrupt method.*
- void [LPUART\\_TransferAbortReceive](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle)  
*Aborts the interrupt-driven data receiving.*

- **status\_t LPUART\_TransferGetReceiveCount** (LPUART\_Type \*base, lpuart\_handle\_t \*handle, uint32\_t \*count)

*Gets the number of bytes that have been received.*

- **void LPUART\_TransferHandleIRQ** (LPUART\_Type \*base, void \*irqHandle)

*LPUART IRQ handle function.*

- **void LPUART\_TransferHandleErrorIRQ** (LPUART\_Type \*base, void \*irqHandle)

*LPUART Error IRQ handle function.*

## 21.2.3 Data Structure Documentation

### 21.2.3.1 struct lpuart\_config\_t

#### Data Fields

- **uint32\_t baudRate\_Bps**  
*LPUART baud rate.*
- **lpuart\_parity\_mode\_t parityMode**  
*Parity mode, disabled (default), even, odd.*
- **lpuart\_data\_bits\_t dataBitsCount**  
*Data bits count, eight (default), seven.*
- **bool isMsb**  
*Data bits order, LSB (default), MSB.*
- **lpuart\_stop\_bit\_count\_t stopBitCount**  
*Number of stop bits, 1 stop bit (default) or 2 stop bits.*
- **bool enableRxRTS**  
*RX RTS enable.*
- **bool enableTxCTS**  
*TX CTS enable.*
- **lpuart\_transmit\_cts\_source\_t txCtsSource**  
*TX CTS source.*
- **lpuart\_transmit\_cts\_config\_t txCtsConfig**  
*TX CTS configure.*
- **lpuart\_idle\_type\_select\_t rxIdleType**  
*RX IDLE type.*
- **lpuart\_idle\_config\_t rxIdleConfig**  
*RX IDLE configuration.*
- **bool enableTx**  
*Enable TX.*
- **bool enableRx**  
*Enable RX.*

#### Field Documentation

- (1) **lpuart\_idle\_type\_select\_t lpuart\_config\_t::rxIdleType**
- (2) **lpuart\_idle\_config\_t lpuart\_config\_t::rxIdleConfig**

### 21.2.3.2 struct lpuart\_transfer\_t

#### Data Fields

- `size_t dataSize`  
*The byte count to be transfer.*
- `uint8_t * data`  
*The buffer of data to be transfer.*
- `uint8_t * rxData`  
*The buffer to receive data.*
- `const uint8_t * txData`  
*The buffer of data to be sent.*

#### Field Documentation

- (1) `uint8_t* lpuart_transfer_t::data`
- (2) `uint8_t* lpuart_transfer_t::rxData`
- (3) `const uint8_t* lpuart_transfer_t::txData`
- (4) `size_t lpuart_transfer_t::dataSize`

### 21.2.3.3 struct \_lpuart\_handle

#### Data Fields

- `const 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.*
- `lpuart_transfer_callback_t callback`  
*Callback function.*
- `void * userData`  
*LPUART callback function parameter.*

- volatile uint8\_t **txState**  
*TX transfer state.*
- volatile uint8\_t **rxState**  
*RX transfer state.*

### Field Documentation

- (1) const uint8\_t\* volatile lpuart\_handle\_t::txData
- (2) volatile size\_t lpuart\_handle\_t::txDataSize
- (3) size\_t lpuart\_handle\_t::txDataSizeAll
- (4) uint8\_t\* volatile lpuart\_handle\_t::rxData
- (5) volatile size\_t lpuart\_handle\_t::rxDataSize
- (6) size\_t lpuart\_handle\_t::rxDataSizeAll
- (7) uint8\_t\* lpuart\_handle\_t::rxRingBuffer
- (8) size\_t lpuart\_handle\_t::rxRingBufferSize
- (9) volatile uint16\_t lpuart\_handle\_t::rxRingBufferHead
- (10) volatile uint16\_t lpuart\_handle\_t::rxRingBufferTail
- (11) lpuart\_transfer\_callback\_t lpuart\_handle\_t::callback
- (12) void\* lpuart\_handle\_t::userData
- (13) volatile uint8\_t lpuart\_handle\_t::txState
- (14) volatile uint8\_t lpuart\_handle\_t::rxState

### 21.2.4 Macro Definition Documentation

21.2.4.1 #define FSL\_LPUART\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 3))

21.2.4.2 #define UART\_RETRY\_TIMES 0U /\* Defining to zero means to keep waiting for the flag until it is assert/deassert. \*/

### 21.2.5 Typedef Documentation

21.2.5.1 typedef void(\* lpuart\_transfer\_callback\_t)(LPUART\_Type \*base, lpuart\_handle\_t \*handle, status\_t status, void \*userData)

## 21.2.6 Enumeration Type Documentation

### 21.2.6.1 anonymous enum

Enumerator

*kStatus\_LPUART\_TxBusy* TX busy.  
*kStatus\_LPUART\_RxBusy* RX busy.  
*kStatus\_LPUART\_TxIdle* LPUART transmitter is idle.  
*kStatus\_LPUART\_RxIdle* LPUART receiver is idle.  
*kStatus\_LPUART\_TxWatermarkTooLarge* TX FIFO watermark too large.  
*kStatus\_LPUART\_RxWatermarkTooLarge* RX FIFO watermark too large.  
*kStatus\_LPUART\_FlagCannotClearManually* Some flag can't manually clear.  
*kStatus\_LPUART\_Error* Error happens on LPUART.  
*kStatus\_LPUART\_RxRingBufferOverrun* LPUART RX software ring buffer overrun.  
*kStatus\_LPUART\_RxHardwareOverrun* LPUART RX receiver overrun.  
*kStatus\_LPUART\_NoiseError* LPUART noise error.  
*kStatus\_LPUART\_FramingError* LPUART framing error.  
*kStatus\_LPUART\_ParityError* LPUART parity error.  
*kStatus\_LPUART\_BaudrateNotSupport* Baudrate is not support in current clock source.  
*kStatus\_LPUART\_IdleLineDetected* IDLE flag.  
*kStatus\_LPUART\_Timeout* LPUART times out.

### 21.2.6.2 enum lpuart\_parity\_mode\_t

Enumerator

*kLPUART\_ParityDisabled* Parity disabled.  
*kLPUART\_ParityEven* Parity enabled, type even, bit setting: PE|PT = 10.  
*kLPUART\_ParityOdd* Parity enabled, type odd, bit setting: PE|PT = 11.

### 21.2.6.3 enum lpuart\_data\_bits\_t

Enumerator

*kLPUART\_EightDataBits* Eight data bit.

### 21.2.6.4 enum lpuart\_stop\_bit\_count\_t

Enumerator

*kLPUART\_OneStopBit* One stop bit.  
*kLPUART\_TwoStopBit* Two stop bits.

### 21.2.6.5 enum lpuart\_transmit\_cts\_source\_t

Enumerator

*kLPUART\_CtsSourcePin* CTS resource is the LPUART\_CTS pin.

*kLPUART\_CtsSourceMatchResult* CTS resource is the match result.

### 21.2.6.6 enum lpuart\_transmit\_cts\_config\_t

Enumerator

*kLPUART\_CtsSampleAtStart* CTS input is sampled at the start of each character.

*kLPUART\_CtsSampleAtIdle* CTS input is sampled when the transmitter is idle.

### 21.2.6.7 enum lpuart\_idle\_type\_select\_t

Enumerator

*kLPUART\_IdleTypeStartBit* Start counting after a valid start bit.

*kLPUART\_IdleTypeStopBit* Start counting after a stop bit.

### 21.2.6.8 enum lpuart\_idle\_config\_t

This structure defines the number of idle characters that must be received before the IDLE flag is set.

Enumerator

*kLPUART\_IdleCharacter1* the number of idle characters.

*kLPUART\_IdleCharacter2* the number of idle characters.

*kLPUART\_IdleCharacter4* the number of idle characters.

*kLPUART\_IdleCharacter8* the number of idle characters.

*kLPUART\_IdleCharacter16* the number of idle characters.

*kLPUART\_IdleCharacter32* the number of idle characters.

*kLPUART\_IdleCharacter64* the number of idle characters.

*kLPUART\_IdleCharacter128* the number of idle characters.

### 21.2.6.9 enum \_lpuart\_interrupt\_enable

This structure contains the settings for all LPUART interrupt configurations.

Enumerator

*kLPUART\_LinBreakInterruptEnable* LIN break detect. bit 7

*kLPUART\_RxActiveEdgeInterruptEnable* Receive Active Edge. bit 6  
*kLPUART\_TxDataRegEmptyInterruptEnable* Transmit data register empty. bit 23  
*kLPUART\_TransmissionCompleteInterruptEnable* Transmission complete. bit 22  
*kLPUART\_RxDataRegFullInterruptEnable* Receiver data register full. bit 21  
*kLPUART\_IdleLineInterruptEnable* Idle line. bit 20  
*kLPUART\_RxOverrunInterruptEnable* Receiver Overrun. bit 27  
*kLPUART\_NoiseErrorInterruptEnable* Noise error flag. bit 26  
*kLPUART\_FramingErrorInterruptEnable* Framing error flag. bit 25  
*kLPUART\_ParityErrorInterruptEnable* Parity error flag. bit 24  
*kLPUART\_Match1InterruptEnable* Parity error flag. bit 15  
*kLPUART\_Match2InterruptEnable* Parity error flag. bit 14

#### 21.2.6.10 enum \_lpuart\_flags

This provides constants for the LPUART status flags for use in the LPUART functions.

Enumerator

*kLPUART\_TxDataRegEmptyFlag* Transmit data register empty flag, sets when transmit buffer is empty. bit 23  
*kLPUART\_TransmissionCompleteFlag* Transmission complete flag, sets when transmission activity complete. bit 22  
*kLPUART\_RxDataRegFullFlag* Receive data register full flag, sets when the receive data buffer is full. bit 21  
*kLPUART\_IdleLineFlag* Idle line detect flag, sets when idle line detected. bit 20  
*kLPUART\_RxOverrunFlag* Receive Overrun, sets when new data is received before data is read from receive register. bit 19  
*kLPUART\_NoiseErrorFlag* Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18  
*kLPUART\_FramingErrorFlag* Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17  
*kLPUART\_ParityErrorFlag* If parity enabled, sets upon parity error detection. bit 16  
*kLPUART\_LinBreakFlag* LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31  
*kLPUART\_RxActiveEdgeFlag* Receive pin active edge interrupt flag, sets when active edge detected. bit 30  
*kLPUART\_RxActiveFlag* Receiver Active Flag (RAF), sets at beginning of valid start. bit 24  
*kLPUART\_DataMatch1Flag* The next character to be read from LPUART\_DATA matches MA1. bit 15  
*kLPUART\_DataMatch2Flag* The next character to be read from LPUART\_DATA matches MA2. bit 14

#### 21.2.7 Function Documentation

### 21.2.7.1 status\_t LPUART\_Init ( LPUART\_Type \* *base*, const lpuart\_config\_t \* *config*, uint32\_t *srcClock\_Hz* )

This function configures the LPUART module with user-defined settings. Call the [LPUART\\_GetDefaultConfig\(\)](#) function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
* lpuart_config_t lpuartConfig;
* lpuartConfig.baudRate_Bps = 115200U;
* lpuartConfig.parityMode = kLPUART_ParityDisabled;
* lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
* lpuartConfig.isMsb = false;
* lpuartConfig.stopBitCount = kLPUART_OneStopBit;
* lpuartConfig.txFifoWatermark = 0;
* lpuartConfig.rxFifoWatermark = 1;
* LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
*
```

Parameters

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>base</i>        | LPUART peripheral base address.                    |
| <i>config</i>      | Pointer to a user-defined configuration structure. |
| <i>srcClock_Hz</i> | LPUART clock source frequency in HZ.               |

Return values

|                                          |                                                  |
|------------------------------------------|--------------------------------------------------|
| <i>kStatus_LPUART_BaudrateNotSupport</i> | Baudrate is not support in current clock source. |
| <i>kStatus_Success</i>                   | LPUART initialize succeed                        |

### 21.2.7.2 void LPUART\_Deinit ( LPUART\_Type \* *base* )

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

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

### 21.2.7.3 void LPUART\_GetDefaultConfig ( lpuart\_config\_t \* *config* )

This function initializes the LPUART configuration structure to a default value. The default values are:  
*: lpuartConfig->baudRate\_Bps = 115200U; lpuartConfig->parityMode = kLPUART\_ParityDisabled;*  
*lpuartConfig->dataBitsCount = kLPUART\_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART\_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART\_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART\_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;*

Parameters

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

#### 21.2.7.4 status\_t LPUART\_SetBaudRate ( LPUART\_Type \* *base*, uint32\_t *baudRate\_Bps*, uint32\_t *srcClock\_Hz* )

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART\_Init.

```
* LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
*
```

Parameters

|                     |                                      |
|---------------------|--------------------------------------|
| <i>base</i>         | LPUART peripheral base address.      |
| <i>baudRate_Bps</i> | LPUART baudrate to be set.           |
| <i>srcClock_Hz</i>  | LPUART clock source frequency in HZ. |

Return values

|                                          |                                                        |
|------------------------------------------|--------------------------------------------------------|
| <i>kStatus_LPUART_BaudrateNotSupport</i> | Baudrate is not supported in the current clock source. |
| <i>kStatus_Success</i>                   | Set baudrate succeeded.                                |

#### 21.2.7.5 void LPUART\_Enable9bitMode ( LPUART\_Type \* *base*, bool *enable* )

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPUART peripheral base address.   |
| <i>enable</i> | true to enable, flase to disable. |

#### 21.2.7.6 static void LPUART\_SetMatchAddress ( LPUART\_Type \* *base*, uint16\_t *address1*, uint16\_t *address2* ) [inline], [static]

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it

receives with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

#### Note

Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

#### Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>base</i>     | LPUART peripheral base address. |
| <i>address1</i> | LPUART slave address1.          |
| <i>address2</i> | LPUART slave address2.          |

### 21.2.7.7 static void LPUART\_EnableMatchAddress ( LPUART\_Type \* *base*, bool *match1*, bool *match2* ) [inline], [static]

#### Parameters

|               |                                                  |
|---------------|--------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                  |
| <i>match1</i> | true to enable match address1, false to disable. |
| <i>match2</i> | true to enable match address2, false to disable. |

### 21.2.7.8 uint32\_t LPUART\_GetStatusFlags ( LPUART\_Type \* *base* )

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators [\\_lpuart\\_flags](#). To check for a specific status, compare the return value with enumerators in the [\\_lpuart\\_flags](#). For example, to check whether the TX is empty:

```
* if (kLPUART_TxDataRegEmptyFlag &
* LPUART_GetStatusFlags(LPUART1))
* {
* ...
* }
```

Parameters

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

Returns

LPUART status flags which are ORed by the enumerators in the \_lpuart\_flags.

### 21.2.7.9 **status\_t LPUART\_ClearStatusFlags ( LPUART\_Type \* *base*, uint32\_t *mask* )**

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only cleared or set by hardware are: kLPUART\_TxDataRegEmptyFlag, kLPUART\_TransmissionCompleteFlag, kLPUART\_RxDataRegFullFlag, kLPUART\_RxActiveFlag, kLPUART\_NoiseErrorFlag, kLPUART\_ParityErrorFlag, kLPUART\_TxFifoEmptyFlag, kLPUART\_RxFifoEmptyFlag Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

|             |                                                                                                                                        |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | LPUART peripheral base address.                                                                                                        |
| <i>mask</i> | the status flags to be cleared. The user can use the enumerators in the _lpuart_status_flag_t to do the OR operation and get the mask. |

Returns

0 succeed, others failed.

Return values

|                                               |                                                                                         |
|-----------------------------------------------|-----------------------------------------------------------------------------------------|
| <i>kStatus_LPUART_FlagCannotClearManually</i> | The flag can't be cleared by this function but it is cleared automatically by hardware. |
| <i>kStatus_Success</i>                        | Status in the mask are cleared.                                                         |

### 21.2.7.10 **void LPUART\_EnableInterrupts ( LPUART\_Type \* *base*, uint32\_t *mask* )**

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the [\\_lpuart\\_interrupt\\_enable](#). This examples shows how to enable TX empty interrupt and RX full interrupt:

```
* LPUART_EnableInterrupts(LPUART1,
 kLPUART_TxDataRegEmptyInterruptEnable |
 kLPUART_RxDataRegFullInterruptEnable);
*
```

## Parameters

|             |                                                                                    |
|-------------|------------------------------------------------------------------------------------|
| <i>base</i> | LPUART peripheral base address.                                                    |
| <i>mask</i> | The interrupts to enable. Logical OR of <a href="#">_lpuart_interrupt_enable</a> . |

**21.2.7.11 void LPUART\_DisableInterrupts ( LPUART\_Type \* *base*, uint32\_t *mask* )**

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See [\\_lpuart\\_interrupt\\_enable](#). This example shows how to disable the TX empty interrupt and RX full interrupt:

```
* LPUART_DisableInterrupts(LPUART1,
* kLPUART_TxDataRegEmptyInterruptEnable |
* kLPUART_RxDataRegFullInterruptEnable);
*
```

## Parameters

|             |                                                                                     |
|-------------|-------------------------------------------------------------------------------------|
| <i>base</i> | LPUART peripheral base address.                                                     |
| <i>mask</i> | The interrupts to disable. Logical OR of <a href="#">_lpuart_interrupt_enable</a> . |

**21.2.7.12 uint32\_t LPUART\_GetEnabledInterrupts ( LPUART\_Type \* *base* )**

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators [\\_lpuart\\_interrupt\\_enable](#). To check a specific interrupt enable status, compare the return value with enumerators in [\\_lpuart\\_interrupt\\_enable](#). For example, to check whether the TX empty interrupt is enabled:

```
* uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);
*
* if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
* {
* ...
* }
*
```

## Parameters

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

## Returns

LPUART interrupt flags which are logical OR of the enumerators in [\\_lpuart\\_interrupt\\_enable](#).

**21.2.7.13 static uint32\_t LPUART\_GetDataRegisterAddress ( LPUART\_Type \* *base* )  
[inline], [static]**

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

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

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

#### 21.2.7.14 static void LPUART\_EnableTxDMA ( LPUART\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPUART peripheral base address.   |
| <i>enable</i> | True to enable, false to disable. |

#### 21.2.7.15 static void LPUART\_EnableRxDMA ( LPUART\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPUART peripheral base address.   |
| <i>enable</i> | True to enable, false to disable. |

#### 21.2.7.16 uint32\_t LPUART\_GetInstance ( LPUART\_Type \* *base* )

Parameters

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

Returns

LPUART instance.

**21.2.7.17 static void LPUART\_EnableTx ( LPUART\_Type \* *base*, bool *enable* )  
[inline], [static]**

This function enables or disables the LPUART transmitter.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPUART peripheral base address.   |
| <i>enable</i> | True to enable, false to disable. |

#### 21.2.7.18 static void LPUART\_EnableRx ( LPUART\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the LPUART receiver.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPUART peripheral base address.   |
| <i>enable</i> | True to enable, false to disable. |

#### 21.2.7.19 static void LPUART\_WriteByte ( LPUART\_Type \* *base*, uint8\_t *data* ) [inline], [static]

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | LPUART peripheral base address. |
| <i>data</i> | Data write to the TX register.  |

#### 21.2.7.20 static uint8\_t LPUART\_ReadByte ( LPUART\_Type \* *base* ) [inline], [static]

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

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

Returns

Data read from data register.

#### 21.2.7.21 void LPUART\_SendAddress ( LPUART\_Type \* *base*, uint8\_t *address* )

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | LPUART peripheral base address. |
| <i>address</i> | LPUART slave address.           |

### 21.2.7.22 status\_t LPUART\_WriteBlocking ( LPUART\_Type \* *base*, const uint8\_t \* *data*, size\_t *length* )

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the dat to be sent out to the bus.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | LPUART peripheral base address.     |
| <i>data</i>   | Start address of the data to write. |
| <i>length</i> | Size of the data to write.          |

Return values

|                                |                                         |
|--------------------------------|-----------------------------------------|
| <i>kStatus_LPUART_-Timeout</i> | Transmission timed out and was aborted. |
| <i>kStatus_Success</i>         | Successfully wrote all data.            |

### 21.2.7.23 status\_t LPUART\_ReadBlocking ( LPUART\_Type \* *base*, uint8\_t \* *data*, size\_t *length* )

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | LPUART 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_LPUART_Rx-HardwareOverrun</i> | Receiver overrun happened while receiving data. |
| <i>kStatus_LPUART_Noise-Error</i>        | Noise error happened while receiving data.      |
| <i>kStatus_LPUART_FramingError</i>       | Framing error happened while receiving data.    |
| <i>kStatus_LPUART_Parity-Error</i>       | Parity error happened while receiving data.     |
| <i>kStatus_LPUART_Timeout</i>            | Transmission timed out and was aborted.         |
| <i>kStatus_Success</i>                   | Successfully received all data.                 |

#### 21.2.7.24 void LPUART\_TransferCreateHandle ( **LPUART\_Type** \* *base*, **Ipuart\_handle\_t** \* *handle*, **Ipuart\_transfer\_callback\_t** *callback*, **void** \* *userData* )

This function initializes the LPUART handle, which can be used for other LPUART transactional APIs. Usually, for a specified LPUART instance, call this API once to get the initialized handle.

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the [LPUART\\_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as *ringBuffer*.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>base</i>     | LPUART peripheral base address. |
| <i>handle</i>   | LPUART handle pointer.          |
| <i>callback</i> | Callback function.              |
| <i>userData</i> | User data.                      |

#### 21.2.7.25 **status\_t** LPUART\_TransferSendNonBlocking ( **LPUART\_Type** \* *base*, **Ipuart\_handle\_t** \* *handle*, **Ipuart\_transfer\_t** \* *xfer* )

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the [kStatus\\_LPUART\\_TxIdle](#) as status parameter.

## Note

The kStatus\_LPUART\_TxIdle is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the T-X, check the kLPUART\_TransmissionCompleteFlag to ensure that the transmit is finished.

## Parameters

|               |                                                                    |
|---------------|--------------------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                                    |
| <i>handle</i> | LPUART handle pointer.                                             |
| <i>xfer</i>   | LPUART transfer structure, see <a href="#">lpuart_transfer_t</a> . |

## Return values

|                                |                                                                                    |
|--------------------------------|------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start the data transmission.                                          |
| <i>kStatus_LPUART_TxBusy</i>   | Previous transmission still not finished, data not all written to the TX register. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.                                                                  |

### 21.2.7.26 void LPUART\_TransferStartRingBuffer ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle*, uint8\_t \* *ringBuffer*, size\_t *ringBufferSize* )

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the [UART\\_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 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>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

|                       |                                                                                              |
|-----------------------|----------------------------------------------------------------------------------------------|
| <i>ringBuffer</i>     | Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer. |
| <i>ringBufferSize</i> | size of the ring buffer.                                                                     |

### 21.2.7.27 void LPUART\_TransferStopRingBuffer ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle* )

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

### 21.2.7.28 size\_t LPUART\_TransferGetRxRingBufferLength ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle* )

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

Returns

Length of received data in RX ring buffer.

### 21.2.7.29 void LPUART\_TransferAbortSend ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle* )

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are not sent out.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

### 21.2.7.30 status\_t LPUART\_TransferGetSendCount ( **LPUART\_Type** \* *base*,                   **Ipuart\_Handle\_t** \* *handle*, **uint32\_t** \* *count* )

This function gets the number of bytes that have been sent out to bus by an interrupt method.

## Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART 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> ; |

**21.2.7.31 status\_t LPUART\_TransferReceiveNonBlocking ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle*, lpuart\_transfer\_t \* *xfer*, size\_t \* *receivedBytes* )**

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are 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 for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter *kStatus\_UART\_RxIdle*. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to *xfer->data*, which returns with the parameter *receivedBytes* set to 5. For the remaining 5 bytes, the newly arrived data is saved from *xfer->data[5]*. When 5 bytes are received, the LPUART 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 *xfer->data*. When all data is received, the upper layer is notified.

## Parameters

|                      |                                                                  |
|----------------------|------------------------------------------------------------------|
| <i>base</i>          | LPUART peripheral base address.                                  |
| <i>handle</i>        | LPUART handle pointer.                                           |
| <i>xfer</i>          | LPUART transfer structure, see <a href="#">uart_transfer_t</a> . |
| <i>receivedBytes</i> | Bytes received from the ring buffer directly.                    |

## Return values

|                                |                                                          |
|--------------------------------|----------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully queue the transfer into the transmit queue. |
| <i>kStatus_LPUART_Rx-Busy</i>  | Previous receive request is not finished.                |
| <i>kStatus_InvalidArgument</i> | Invalid argument.                                        |

### 21.2.7.32 void LPUART\_TransferAbortReceive ( LPUART\_Type \* *base*, Ipuart\_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>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

### 21.2.7.33 status\_t LPUART\_TransferGetReceiveCount ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been received.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |
| <i>count</i>  | Receive bytes count.            |

Return values

|                                      |                                                       |
|--------------------------------------|-------------------------------------------------------|
| <i>kStatus_NoTransferIn-Progress</i> | No receive in progress.                               |
| <i>kStatus_InvalidArgument</i>       | Parameter is invalid.                                 |
| <i>kStatus_Success</i>               | Get successfully through the parameter <i>count</i> ; |

### 21.2.7.34 void LPUART\_TransferHandleIRQ ( LPUART\_Type \* *base*, void \* *irqHandle* )

This function handles the LPUART transmit and receive IRQ request.

Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>base</i>      | LPUART peripheral base address. |
| <i>irqHandle</i> | LPUART handle pointer.          |

### 21.2.7.35 void LPUART\_TransferHandleErrorIRQ ( LPUART\_Type \* *base*, void \* *irqHandle* )

This function handles the LPUART error IRQ request.

Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>base</i>      | LPUART peripheral base address. |
| <i>irqHandle</i> | LPUART handle pointer.          |

## 21.3 LPUART eDMA Driver

### 21.3.1 Overview

#### Data Structures

- struct [lpuart\\_edma\\_handle\\_t](#)  
*LPUART eDMA handle.* [More...](#)

#### TypeDefs

- [typedef void\(\\* lpuart\\_edma\\_transfer\\_callback\\_t \)](#)(LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*LPUART transfer callback function.*

#### Driver version

- #define [FSL\\_LPUART\\_EDMA\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 5, 2))  
*LPUART EDMA driver version.*

#### eDMA transactional

- void [LPUART\\_TransferCreateHandleEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, [lpuart\\_edma\\_transfer\\_callback\\_t](#) callback, void \*userData, [edma\\_handle\\_t](#) \*txEdmaHandle, [edma\\_handle\\_t](#) \*rxEdmaHandle)  
*Initializes the LPUART handle which is used in transactional functions.*
- [status\\_t LPUART\\_SendEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, [lpuart\\_transfer\\_t](#) \*xfer)  
*Sends data using eDMA.*
- [status\\_t LPUART\\_ReceiveEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, [lpuart\\_transfer\\_t](#) \*xfer)  
*Receives data using eDMA.*
- void [LPUART\\_TransferAbortSendEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle)  
*Aborts the sent data using eDMA.*
- void [LPUART\\_TransferAbortReceiveEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle)  
*Aborts the received data using eDMA.*
- [status\\_t LPUART\\_TransferGetSendCountEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of bytes written to the LPUART TX register.*
- [status\\_t LPUART\\_TransferGetReceiveCountEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of received bytes.*
- void [LPUART\\_TransferEdmaHandleIRQ](#) (LPUART\_Type \*base, void \*lpuartEdmaHandle)  
*LPUART eDMA IRQ handle function.*

## 21.3.2 Data Structure Documentation

### 21.3.2.1 struct \_lpuart\_edma\_handle

#### Data Fields

- `lpuart_edma_transfer_callback_t callback`  
*Callback function.*
- `void *userData`  
*LPUART callback function parameter.*
- `size_t rxDataSizeAll`  
*Size of the data to receive.*
- `size_t txDataSizeAll`  
*Size of the data to send out.*
- `edma_handle_t *txEdmaHandle`  
*The eDMA TX channel used.*
- `edma_handle_t *rxEdmaHandle`  
*The eDMA RX channel used.*
- `uint8_t nbytes`  
*eDMA minor byte transfer count initially configured.*
- `volatile uint8_t txState`  
*TX transfer state.*
- `volatile uint8_t rxState`  
*RX transfer state.*

#### Field Documentation

- (1) `lpuart_edma_transfer_callback_t lpuart_edma_handle_t::callback`
- (2) `void* lpuart_edma_handle_t::userData`
- (3) `size_t lpuart_edma_handle_t::rxDataSizeAll`
- (4) `size_t lpuart_edma_handle_t::txDataSizeAll`
- (5) `edma_handle_t* lpuart_edma_handle_t::txEdmaHandle`
- (6) `edma_handle_t* lpuart_edma_handle_t::rxEdmaHandle`
- (7) `uint8_t lpuart_edma_handle_t::nbytes`
- (8) `volatile uint8_t lpuart_edma_handle_t::txState`

## 21.3.3 Macro Definition Documentation

### 21.3.3.1 #define FSL\_LPUART\_EDMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 2))

## 21.3.4 Typedef Documentation

**21.3.4.1 `typedef void(* lpuart_edma_transfer_callback_t)(LPUART_Type *base, lpuart_edma_handle_t *handle, status_t status, void *userData)`**

### 21.3.5 Function Documentation

**21.3.5.1 `void LPUART_TransferCreateHandleEDMA ( LPUART_Type * base, lpuart_edma_handle_t * handle, lpuart_edma_transfer_callback_t callback, void * userData, edma_handle_t * txEdmaHandle, edma_handle_t * rxEdmaHandle )`**

#### Note

This function disables all LPUART interrupts.

#### Parameters

|                     |                                                         |
|---------------------|---------------------------------------------------------|
| <i>base</i>         | LPUART peripheral base address.                         |
| <i>handle</i>       | Pointer to <code>lpuart_edma_handle_t</code> structure. |
| <i>callback</i>     | Callback function.                                      |
| <i>userData</i>     | User data.                                              |
| <i>txEdmaHandle</i> | User requested DMA handle for TX DMA transfer.          |
| <i>rxEdmaHandle</i> | User requested DMA handle for RX DMA transfer.          |

**21.3.5.2 `status_t LPUART_SendEDMA ( LPUART_Type * base, lpuart_edma_handle_t * handle, lpuart_transfer_t * xfer )`**

This function sends data using eDMA. 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>   | LPUART peripheral base address.                                         |
| <i>handle</i> | LPUART handle pointer.                                                  |
| <i>xfer</i>   | LPUART eDMA transfer structure. See <a href="#">lpuart_transfer_t</a> . |

#### Return values

---

|                                |                             |
|--------------------------------|-----------------------------|
| <i>kStatus_Success</i>         | if succeed, others failed.  |
| <i>kStatus_LPUART_TxBusy</i>   | Previous transfer on going. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.           |

### 21.3.5.3 status\_t LPUART\_ReceiveEDMA ( LPUART\_Type \* *base*, Ipuart\_edma\_handle\_t \* *handle*, Ipuart\_transfer\_t \* *xfer* )

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

|               |                                                                         |
|---------------|-------------------------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                                         |
| <i>handle</i> | Pointer to Ipuart_edma_handle_t structure.                              |
| <i>xfer</i>   | LPUART eDMA transfer structure, see <a href="#">Ipuart_transfer_t</a> . |

Return values

|                                |                            |
|--------------------------------|----------------------------|
| <i>kStatus_Success</i>         | if succeed, others fail.   |
| <i>kStatus_LPUART_Rx-Busy</i>  | Previous transfer ongoing. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.          |

### 21.3.5.4 void LPUART\_TransferAbortSendEDMA ( LPUART\_Type \* *base*, Ipuart\_edma\_handle\_t \* *handle* )

This function aborts the sent data using eDMA.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.            |
| <i>handle</i> | Pointer to Ipuart_edma_handle_t structure. |

### 21.3.5.5 void LPUART\_TransferAbortReceiveEDMA ( LPUART\_Type \* *base*, Ipuart\_edma\_handle\_t \* *handle* )

This function aborts the received data using eDMA.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.            |
| <i>handle</i> | Pointer to lpuart_edma_handle_t structure. |

### 21.3.5.6 status\_t LPUART\_TransferGetSendCountEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART 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> ; |

### 21.3.5.7 status\_t LPUART\_TransferGetReceiveCountEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of received bytes.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART 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; |

#### 21.3.5.8 **void LPUART\_TransferEdmaHandleIRQ ( LPUART\_Type \* *base*, void \* *lpuartEdmaHandle* )**

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

Note

This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

|                         |                                 |
|-------------------------|---------------------------------|
| <i>base</i>             | LPUART peripheral base address. |
| <i>lpuartEdmaHandle</i> | LPUART handle pointer.          |

## 21.4 LPUART FreeRTOS Driver

### 21.4.1 Overview

#### Data Structures

- struct `lpuart_rtos_config_t`  
*LPUART RTOS configuration structure.* [More...](#)

#### Driver version

- #define `FSL_LPUART_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 6, 0)`)  
*LPUART FreeRTOS driver version.*

#### LPUART RTOS Operation

- int `LPUART_RTOS_Init` (`lpuart_rtos_handle_t *handle, lpuart_handle_t *t_handle, const lpuart_rtos_config_t *cfg`)  
*Initializes an LPUART instance for operation in RTOS.*
- int `LPUART_RTOS_Deinit` (`lpuart_rtos_handle_t *handle`)  
*Deinitializes an LPUART instance for operation.*

#### LPUART transactional Operation

- int `LPUART_RTOS_Send` (`lpuart_rtos_handle_t *handle, uint8_t *buffer, uint32_t length`)  
*Sends data in the background.*
- int `LPUART_RTOS_Receive` (`lpuart_rtos_handle_t *handle, uint8_t *buffer, uint32_t length, size_t *received`)  
*Receives data.*
- int `LPUART_RTOS_SetRxTimeout` (`lpuart_rtos_handle_t *handle, uint32_t rx_timeout_constant_ms, uint32_t rx_timeout_multiplier_ms`)  
*Set RX timeout in runtime.*
- int `LPUART_RTOS_SetTxTimeout` (`lpuart_rtos_handle_t *handle, uint32_t tx_timeout_constant_ms, uint32_t tx_timeout_multiplier_ms`)  
*Set TX timeout in runtime.*

### 21.4.2 Data Structure Documentation

#### 21.4.2.1 struct `lpuart_rtos_config_t`

##### Data Fields

- `LPUART_Type * base`  
*UART base address.*

- `uint32_t srclk`  
*UART source clock in Hz.*
- `uint32_t baudrate`  
*Desired communication speed.*
- `lpuart_parity_mode_t parity`  
*Parity setting.*
- `lpuart_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.*
- `uint32_t rx_timeout_constant_ms`  
*RX timeout applied per receive.*
- `uint32_t rx_timeout_multiplier_ms`  
*RX timeout added for each byte of the receive.*
- `uint32_t tx_timeout_constant_ms`  
*TX timeout applied per transmission.*
- `uint32_t tx_timeout_multiplier_ms`  
*TX timeout added for each byte of the transmission.*
- `bool enableRxRTS`  
*RX RTS enable.*
- `bool enableTxCTS`  
*TX CTS enable.*
- `lpuart_transmit_cts_source_t txCtsSource`  
*TX CTS source.*
- `lpuart_transmit_cts_config_t txCtsConfig`  
*TX CTS configure.*

## Field Documentation

- (1) `uint32_t lpuart_rtos_config_t::rx_timeout_multiplier_ms`
- (2) `uint32_t lpuart_rtos_config_t::tx_timeout_multiplier_ms`

## 21.4.3 Macro Definition Documentation

**21.4.3.1 #define FSL\_LPUART\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 6, 0))**

## 21.4.4 Function Documentation

**21.4.4.1 int LPUART\_RTOS\_Init ( `lpuart_rtos_handle_t * handle`, `lpuart_handle_t * t_handle`, `const lpuart_rtos_config_t * cfg` )**

Parameters

|                 |                                                                                      |
|-----------------|--------------------------------------------------------------------------------------|
| <i>handle</i>   | The RTOS LPUART handle, the pointer to an allocated space for RTOS context.          |
| <i>t_handle</i> | The pointer to an allocated space to store the transactional layer internal state.   |
| <i>cfg</i>      | The pointer to the parameters required to configure the LPUART after initialization. |

Returns

0 succeed, others failed

#### 21.4.4.2 int LPUART\_RTOS\_Deinit ( *Ipuart\_rtos\_handle\_t \* handle* )

This function deinitializes the LPUART module, sets all register value to the reset value, and releases the resources.

Parameters

|               |                         |
|---------------|-------------------------|
| <i>handle</i> | The RTOS LPUART handle. |
|---------------|-------------------------|

#### 21.4.4.3 int LPUART\_RTOS\_Send ( *Ipuart\_rtos\_handle\_t \* handle, uint8\_t \* buffer, uint32\_t length* )

This function sends data. It is an synchronous API. If the hardware buffer is full, the task is in the blocked state.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>handle</i> | The RTOS LPUART handle.        |
| <i>buffer</i> | The pointer to buffer to send. |
| <i>length</i> | The number of bytes to send.   |

#### 21.4.4.4 int LPUART\_RTOS\_Receive ( *Ipuart\_rtos\_handle\_t \* handle, uint8\_t \* buffer, uint32\_t length, size\_t \* received* )

This function receives data from LPUART. It is an synchronous API. If any data is immediately available it is returned immediately and the number of bytes received.

Parameters

|                 |                                                                                  |
|-----------------|----------------------------------------------------------------------------------|
| <i>handle</i>   | The RTOS LPUART 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. |

#### 21.4.4.5 int LPUART\_RTOSETXTIMEOUT ( Ipuart\_rtos\_handle\_t \* *handle*, uint32\_t *rx\_timeout\_constant\_ms*, uint32\_t *rx\_timeout\_multiplier\_ms* )

This function can modify RX timeout between initialization and receive.

param handle The RTOS LPUART handle. param rx\_timeout\_constant\_ms RX timeout applied per receive. param rx\_timeout\_multiplier\_ms RX timeout added for each byte of the receive.

#### 21.4.4.6 int LPUART\_RTOSETXTIMEOUT ( Ipuart\_rtos\_handle\_t \* *handle*, uint32\_t *tx\_timeout\_constant\_ms*, uint32\_t *tx\_timeout\_multiplier\_ms* )

This function can modify TX timeout between initialization and send.

param handle The RTOS LPUART handle. param tx\_timeout\_constant\_ms TX timeout applied per transmission. param tx\_timeout\_multiplier\_ms TX timeout added for each byte of the transmission.

## 21.5 LPUART CMSIS Driver

This section describes the programming interface of the LPUART Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method please refer to <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

The LPUART driver includes transactional APIs.

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 please write custom code.

### 21.5.1 Function groups

#### 21.5.1.1 LPUART CMSIS GetVersion Operation

This function group will return the LPUART CMSIS Driver version to user.

#### 21.5.1.2 LPUART CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

#### 21.5.1.3 LPUART CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialized the lpuart instance . And this API must be called before you configure a lpuart instance or after you Deinit a lpuart instance.The right steps to start an instance is that you must initialize the instance which been selected firstly,then you can power on the instance.After these all have been done,you can configure the instance by using control operation.If you want to Uninitialize the instance, you must power off the instance first.

#### 21.5.1.4 LPUART CMSIS Transfer Operation

This function group controls the transfer, send/receive data.

#### 21.5.1.5 LPUART CMSIS Status Operation

This function group gets the LPUART transfer status.

### **21.5.1.6 LPUART CMSIS Control Operation**

This function can configure an instance ,set baudrate for lpuart, get current baudrate ,set transfer data bits and other control command.

# Chapter 22

## PDB: Programmable Delay Block

### 22.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Programmable Delay Block (PDB) module of MCUXpresso SDK devices.

The PDB driver includes a basic PDB counter, trigger generators for ADC, DAC, and pulse-out.

The basic PDB counter can be used as a general programmable timer with an interrupt. The counter increases automatically with the divided clock signal after it is triggered to start by an external trigger input or the software trigger. There are "milestones" for the output trigger event. When the counter is equal to any of these "milestones", the corresponding trigger is generated and sent out to other modules. These "milestones" are for the following events.

- Counter delay interrupt, which is the interrupt for the PDB module
- ADC pre-trigger to trigger the ADC conversion
- DAC interval trigger to trigger the DAC buffer and move the buffer read pointer
- Pulse-out triggers to generate a single or rising and falling edges, which can be assembled to a window.

The "milestone" values have a flexible load mode. To call the APIs to set these value is equivalent to writing data to their buffer. The loading event occurs as the load mode describes. This design ensures that all "milestones" can be updated at the same time.

### 22.2 Typical use case

#### 22.2.1 Working as basic PDB counter with a PDB interrupt.

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/pdb

#### 22.2.2 Working with an additional trigger. The ADC trigger is used as an example.

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/pdb

### Data Structures

- struct [pdb\\_config\\_t](#)  
*PDB module configuration. [More...](#)*
- struct [pdb\\_adc\\_pretrigger\\_config\\_t](#)  
*PDB ADC Pre-trigger configuration. [More...](#)*
- struct [pdb\\_dac\\_trigger\\_config\\_t](#)  
*PDB DAC trigger configuration. [More...](#)*

## Enumerations

- enum `_pdb_status_flags` {
   
  `kPDB_LoadOKFlag` = `PDB_SC_LDOK_MASK`,
   
  `kPDB_DelayEventFlag` = `PDB_SC_PDBIF_MASK` }
   
*PDB flags.*
- enum `_pdb_adc_pretrigger_flags` {
   
  `kPDB_ADCPreTriggerChannel0Flag` = `PDB_S_CF(1U << 0)`,
   
  `kPDB_ADCPreTriggerChannel1Flag` = `PDB_S_CF(1U << 1)`,
   
  `kPDB_ADCPreTriggerChannel0ErrorFlag` = `PDB_S_ERR(1U << 0)`,
   
  `kPDB_ADCPreTriggerChannel1ErrorFlag` = `PDB_S_ERR(1U << 1)` }
   
*PDB ADC PreTrigger channel flags.*
- enum `_pdb_interrupt_enable` {
   
  `kPDB_SequenceErrorInterruptEnable` = `PDB_SC_PDDEIE_MASK`,
   
  `kPDB_DelayInterruptEnable` = `PDB_SC_PDBIE_MASK` }
   
*PDB buffer interrupts.*
- enum `pdb_load_value_mode_t` {
   
  `kPDB_LoadValueImmediately` = `0U`,
   
  `kPDB_LoadValueOnCounterOverflow` = `1U`,
   
  `kPDB_LoadValueOnTriggerInput` = `2U`,
   
  `kPDB_LoadValueOnCounterOverflowOrTriggerInput` = `3U` }
   
*PDB load value mode.*
- enum `pdb_prescaler_divider_t` {
   
  `kPDB_PrescalerDivider1` = `0U`,
   
  `kPDB_PrescalerDivider2` = `1U`,
   
  `kPDB_PrescalerDivider4` = `2U`,
   
  `kPDB_PrescalerDivider8` = `3U`,
   
  `kPDB_PrescalerDivider16` = `4U`,
   
  `kPDB_PrescalerDivider32` = `5U`,
   
  `kPDB_PrescalerDivider64` = `6U`,
   
  `kPDB_PrescalerDivider128` = `7U` }
   
*Prescaler divider.*
- enum `pdb_divider_multiplication_factor_t` {
   
  `kPDB_DividerMultiplicationFactor1` = `0U`,
   
  `kPDB_DividerMultiplicationFactor10` = `1U`,
   
  `kPDB_DividerMultiplicationFactor20` = `2U`,
   
  `kPDB_DividerMultiplicationFactor40` = `3U` }
   
*Multiplication factor select for prescaler.*
- enum `pdb_trigger_input_source_t` {

```
kPDB_TriggerInput0 = 0U,
kPDB_TriggerInput1 = 1U,
kPDB_TriggerInput2 = 2U,
kPDB_TriggerInput3 = 3U,
kPDB_TriggerInput4 = 4U,
kPDB_TriggerInput5 = 5U,
kPDB_TriggerInput6 = 6U,
kPDB_TriggerInput7 = 7U,
kPDB_TriggerInput8 = 8U,
kPDB_TriggerInput9 = 9U,
kPDB_TriggerInput10 = 10U,
kPDB_TriggerInput11 = 11U,
kPDB_TriggerInput12 = 12U,
kPDB_TriggerInput13 = 13U,
kPDB_TriggerInput14 = 14U,
kPDB_TriggerSoftware = 15U }
```

*Trigger input source.*

- enum `pdb_adc_trigger_channel_t` {
 

```
kPDB_ADCTriggerChannel0 = 0U,
kPDB_ADCTriggerChannel1 = 1U,
kPDB_ADCTriggerChannel2 = 2U,
kPDB_ADCTriggerChannel3 = 3U }
```

*List of PDB ADC trigger channels.*

- enum `pdb_adc_pretrigger_t` {
 

```
kPDB_ADCPreTrigger0 = 0U,
kPDB_ADCPreTrigger1 = 1U,
kPDB_ADCPreTrigger2 = 2U,
kPDB_ADCPreTrigger3 = 3U,
kPDB_ADCPreTrigger4 = 4U,
kPDB_ADCPreTrigger5 = 5U,
kPDB_ADCPreTrigger6 = 6U,
kPDB_ADCPreTrigger7 = 7U }
```

*List of PDB ADC pretrigger.*

- enum `pdb_dac_trigger_channel_t` {
 

```
kPDB_DACTriggerChannel0 = 0U,
kPDB_DACTriggerChannel1 = 1U }
```

*List of PDB DAC trigger channels.*

- enum `pdb_pulse_out_trigger_channel_t` {
 

```
kPDB_PulseOutTriggerChannel0 = 0U,
kPDB_PulseOutTriggerChannel1 = 1U,
kPDB_PulseOutTriggerChannel2 = 2U,
kPDB_PulseOutTriggerChannel3 = 3U }
```

*List of PDB pulse out trigger channels.*

- enum `pdb_pulse_out_channel_mask_t` {

```
kPDB_PulseOutChannel0Mask = (1U << 0U),
kPDB_PulseOutChannel1Mask = (1U << 1U),
kPDB_PulseOutChannel2Mask = (1U << 2U),
kPDB_PulseOutChannel3Mask = (1U << 3U) }
```

*List of PDB pulse out trigger channels mask.*

## Driver version

- #define **FSL\_PDB\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 4))  
*PDB driver version 2.0.4.*

## Initialization

- void **PDB\_Init** (PDB\_Type \*base, const **pdb\_config\_t** \*config)  
*Initializes the PDB module.*
- void **PDB\_Deinit** (PDB\_Type \*base)  
*De-initializes the PDB module.*
- void **PDB\_GetDefaultConfig** (**pdb\_config\_t** \*config)  
*Initializes the PDB user configuration structure.*
- static void **PDB\_Enable** (PDB\_Type \*base, bool enable)  
*Enables the PDB module.*

## Basic Counter

- static void **PDB\_DoSoftwareTrigger** (PDB\_Type \*base)  
*Triggers the PDB counter by software.*
- static void **PDB\_DoLoadValues** (PDB\_Type \*base)  
*Loads the counter values.*
- static void **PDB\_EnableDMA** (PDB\_Type \*base, bool enable)  
*Enables the DMA for the PDB module.*
- static void **PDB\_EnableInterrupts** (PDB\_Type \*base, uint32\_t mask)  
*Enables the interrupts for the PDB module.*
- static void **PDB\_DisableInterrupts** (PDB\_Type \*base, uint32\_t mask)  
*Disables the interrupts for the PDB module.*
- static uint32\_t **PDB\_GetStatusFlags** (PDB\_Type \*base)  
*Gets the status flags of the PDB module.*
- static void **PDB\_ClearStatusFlags** (PDB\_Type \*base, uint32\_t mask)  
*Clears the status flags of the PDB module.*
- static void **PDB\_SetModulusValue** (PDB\_Type \*base, uint32\_t value)  
*Specifies the counter period.*
- static uint32\_t **PDB\_GetCounterValue** (PDB\_Type \*base)  
*Gets the PDB counter's current value.*
- static void **PDB\_SetCounterDelayValue** (PDB\_Type \*base, uint32\_t value)  
*Sets the value for the PDB counter delay event.*

## ADC Pre-trigger

- static void **PDB\_SetADCPreTriggerConfig** (PDB\_Type \*base, **pdb\_adc\_trigger\_channel\_t** channel, **pdb\_adc\_pretrigger\_config\_t** \*config)  
*Configures the ADC pre-trigger in the PDB module.*

- static void `PDB_SetADCPreTriggerDelayValue` (PDB\_Type \*base, `pdb_adc_trigger_channel_t` channel, `pdb_adc_pretrigger_t` pretriggerNumber, uint32\_t value)  
*Sets the value for the ADC pre-trigger delay event.*
- static uint32\_t `PDB_GetADCPreTriggerStatusFlags` (PDB\_Type \*base, `pdb_adc_trigger_channel_t` channel)  
*Gets the ADC pre-trigger's status flags.*
- static void `PDB_ClearADCPreTriggerStatusFlags` (PDB\_Type \*base, `pdb_adc_trigger_channel_t` channel, uint32\_t mask)  
*Clears the ADC pre-trigger status flags.*

## DAC Interval Trigger

- void `PDB_SetDACTriggerConfig` (PDB\_Type \*base, `pdb_dac_trigger_channel_t` channel, `pdb_dac_trigger_config_t` \*config)  
*Configures the DAC trigger in the PDB module.*
- static void `PDB_SetDACTriggerIntervalValue` (PDB\_Type \*base, `pdb_dac_trigger_channel_t` channel, uint32\_t value)  
*Sets the value for the DAC interval event.*

## Pulse-Out Trigger

- static void `PDB_EnablePulseOutTrigger` (PDB\_Type \*base, `pdb_pulse_out_channel_mask_t` channelMask, bool enable)  
*Enables the pulse out trigger channels.*
- static void `PDB_SetPulseOutTriggerDelayValue` (PDB\_Type \*base, `pdb_pulse_out_trigger_channel_t` channel, uint32\_t value1, uint32\_t value2)  
*Sets event values for the pulse out trigger.*

## 22.3 Data Structure Documentation

### 22.3.1 struct `pdb_config_t`

#### Data Fields

- `pdb_load_value_mode_t` `loadValueMode`  
*Select the load value mode.*
- `pdb_prescaler_divider_t` `prescalerDivider`  
*Select the prescaler divider.*
- `pdb_divider_multiplication_factor_t` `dividerMultiplicationFactor`  
*Multiplication factor select for prescaler.*
- `pdb_trigger_input_source_t` `triggerInputSource`  
*Select the trigger input source.*
- bool `enableContinuousMode`  
*Enable the PDB operation in Continuous mode.*

#### Field Documentation

##### (1) `pdb_load_value_mode_t` `pdb_config_t::loadValueMode`

- (2) `pdb_prescaler_divider_t` `pdb_config_t::prescalerDivider`
- (3) `pdb_divider_multiplication_factor_t` `pdb_config_t::dividerMultiplicationFactor`
- (4) `pdb_trigger_input_source_t` `pdb_config_t::triggerInputSource`
- (5) `bool` `pdb_config_t::enableContinuousMode`

### 22.3.2 struct `pdb_adc_pretrigger_config_t`

#### Data Fields

- `uint32_t enablePreTriggerMask`  
*PDB Channel Pre-trigger Enable.*
- `uint32_t enableOutputMask`  
*PDB Channel Pre-trigger Output Select.*
- `uint32_t enableBackToBackOperationMask`  
*PDB Channel pre-trigger Back-to-Back Operation Enable.*

#### Field Documentation

- (1) `uint32_t` `pdb_adc_pretrigger_config_t::enablePreTriggerMask`

- (2) `uint32_t` `pdb_adc_pretrigger_config_t::enableOutputMask`

PDB channel's corresponding pre-trigger asserts when the counter reaches the channel delay register.

- (3) `uint32_t` `pdb_adc_pretrigger_config_t::enableBackToBackOperationMask`

Back-to-back operation enables the ADC conversions complete to trigger the next PDB channel pre-trigger and trigger output, so that the ADC conversions can be triggered on next set of configuration and results registers.

### 22.3.3 struct `pdb_dac_trigger_config_t`

#### Data Fields

- `bool enableExternalTriggerInput`  
*Enables the external trigger for DAC interval counter.*
- `bool enableIntervalTrigger`  
*Enables the DAC interval trigger.*

#### Field Documentation

- (1) `bool` `pdb_dac_trigger_config_t::enableExternalTriggerInput`

- (2) `bool` `pdb_dac_trigger_config_t::enableIntervalTrigger`

## 22.4 Macro Definition Documentation

### 22.4.1 #define FSL\_PDB\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 4))

## 22.5 Enumeration Type Documentation

### 22.5.1 enum \_pdb\_status\_flags

Enumerator

*kPDB\_LoadOKFlag* This flag is automatically cleared when the values in buffers are loaded into the internal registers after the LDOK bit is set or the PDBEN is cleared.

*kPDB\_DelayEventFlag* PDB timer delay event flag.

### 22.5.2 enum \_pdb\_adc\_pretrigger\_flags

Enumerator

*kPDB\_ADCPreTriggerChannel0Flag* Pre-trigger 0 flag.

*kPDB\_ADCPreTriggerChannel1Flag* Pre-trigger 1 flag.

*kPDB\_ADCPreTriggerChannel0ErrorFlag* Pre-trigger 0 Error.

*kPDB\_ADCPreTriggerChannel1ErrorFlag* Pre-trigger 1 Error.

### 22.5.3 enum \_pdb\_interrupt\_enable

Enumerator

*kPDB\_SequenceErrorInterruptEnable* PDB sequence error interrupt enable.

*kPDB\_DelayInterruptEnable* PDB delay interrupt enable.

### 22.5.4 enum pdb\_load\_value\_mode\_t

Selects the mode to load the internal values after doing the load operation (write 1 to PDBx\_SC[LDOK]). These values are for the following operations.

- PDB counter (PDBx\_MOD, PDBx\_IDLY)
- ADC trigger (PDBx\_CHnDLYm)
- DAC trigger (PDBx\_DACINTx)
- CMP trigger (PDBx\_POyDLY)

Enumerator

*kPDB\_LoadValueImmediately* Load immediately after 1 is written to LDOK.

***kPDB\_LoadValueOnCounterOverflow*** Load when the PDB counter overflows (reaches the MOD register value).

***kPDB\_LoadValueOnTriggerInput*** Load a trigger input event is detected.

***kPDB\_LoadValueOnCounterOverflowOrTriggerInput*** Load either when the PDB counter overflows or a trigger input is detected.

## 22.5.5 enum pdb\_prescaler\_divider\_t

Counting uses the peripheral clock divided by multiplication factor selected by times of MULT.

Enumerator

***kPDB\_PrescalerDivider1*** Divider x1.

***kPDB\_PrescalerDivider2*** Divider x2.

***kPDB\_PrescalerDivider4*** Divider x4.

***kPDB\_PrescalerDivider8*** Divider x8.

***kPDB\_PrescalerDivider16*** Divider x16.

***kPDB\_PrescalerDivider32*** Divider x32.

***kPDB\_PrescalerDivider64*** Divider x64.

***kPDB\_PrescalerDivider128*** Divider x128.

## 22.5.6 enum pdb\_divider\_multiplication\_factor\_t

Selects the multiplication factor of the prescaler divider for the counter clock.

Enumerator

***kPDB\_DividerMultiplicationFactor1*** Multiplication factor is 1.

***kPDB\_DividerMultiplicationFactor10*** Multiplication factor is 10.

***kPDB\_DividerMultiplicationFactor20*** Multiplication factor is 20.

***kPDB\_DividerMultiplicationFactor40*** Multiplication factor is 40.

## 22.5.7 enum pdb\_trigger\_input\_source\_t

Selects the trigger input source for the PDB. The trigger input source can be internal or external (EXTRG pin), or the software trigger. See chip configuration details for the actual PDB input trigger connections.

Enumerator

***kPDB\_TriggerInput0*** Trigger-In 0.

***kPDB\_TriggerInput1*** Trigger-In 1.

|                             |                                  |
|-----------------------------|----------------------------------|
| <i>kPDB_TriggerInput2</i>   | Trigger-In 2.                    |
| <i>kPDB_TriggerInput3</i>   | Trigger-In 3.                    |
| <i>kPDB_TriggerInput4</i>   | Trigger-In 4.                    |
| <i>kPDB_TriggerInput5</i>   | Trigger-In 5.                    |
| <i>kPDB_TriggerInput6</i>   | Trigger-In 6.                    |
| <i>kPDB_TriggerInput7</i>   | Trigger-In 7.                    |
| <i>kPDB_TriggerInput8</i>   | Trigger-In 8.                    |
| <i>kPDB_TriggerInput9</i>   | Trigger-In 9.                    |
| <i>kPDB_TriggerInput10</i>  | Trigger-In 10.                   |
| <i>kPDB_TriggerInput11</i>  | Trigger-In 11.                   |
| <i>kPDB_TriggerInput12</i>  | Trigger-In 12.                   |
| <i>kPDB_TriggerInput13</i>  | Trigger-In 13.                   |
| <i>kPDB_TriggerInput14</i>  | Trigger-In 14.                   |
| <i>kPDB_TriggerSoftware</i> | Trigger-In 15, software trigger. |

## 22.5.8 enum pdb\_adc\_trigger\_channel\_t

Note

Actual number of available channels is SoC dependent

Enumerator

|                                |                                   |
|--------------------------------|-----------------------------------|
| <i>kPDB_ADCTriggerChannel0</i> | PDB ADC trigger channel number 0. |
| <i>kPDB_ADCTriggerChannel1</i> | PDB ADC trigger channel number 1. |
| <i>kPDB_ADCTriggerChannel2</i> | PDB ADC trigger channel number 2. |
| <i>kPDB_ADCTriggerChannel3</i> | PDB ADC trigger channel number 3. |

## 22.5.9 enum pdb\_adc\_pretrigger\_t

Note

Actual number of available pretrigger channels is SoC dependent

Enumerator

|                            |                              |
|----------------------------|------------------------------|
| <i>kPDB_ADCPreTrigger0</i> | PDB ADC pretrigger number 0. |
| <i>kPDB_ADCPreTrigger1</i> | PDB ADC pretrigger number 1. |
| <i>kPDB_ADCPreTrigger2</i> | PDB ADC pretrigger number 2. |
| <i>kPDB_ADCPreTrigger3</i> | PDB ADC pretrigger number 3. |
| <i>kPDB_ADCPreTrigger4</i> | PDB ADC pretrigger number 4. |
| <i>kPDB_ADCPreTrigger5</i> | PDB ADC pretrigger number 5. |
| <i>kPDB_ADCPreTrigger6</i> | PDB ADC pretrigger number 6. |
| <i>kPDB_ADCPreTrigger7</i> | PDB ADC pretrigger number 7. |

### 22.5.10 enum pdb\_dac\_trigger\_channel\_t

Note

Actual number of available channels is SoC dependent

Enumerator

**kPDB\_DACTriggerChannel0** PDB DAC trigger channel number 0.  
**kPDB\_DACTriggerChannel1** PDB DAC trigger channel number 1.

### 22.5.11 enum pdb\_pulse\_out\_trigger\_channel\_t

Note

Actual number of available channels is SoC dependent

Enumerator

**kPDB\_PulseOutTriggerChannel0** PDB pulse out trigger channel number 0.  
**kPDB\_PulseOutTriggerChannel1** PDB pulse out trigger channel number 1.  
**kPDB\_PulseOutTriggerChannel2** PDB pulse out trigger channel number 2.  
**kPDB\_PulseOutTriggerChannel3** PDB pulse out trigger channel number 3.

### 22.5.12 enum pdb\_pulse\_out\_channel\_mask\_t

Note

Actual number of available channels mask is SoC dependent

Enumerator

**kPDB\_PulseOutChannel0Mask** PDB pulse out trigger channel number 0 mask.  
**kPDB\_PulseOutChannel1Mask** PDB pulse out trigger channel number 1 mask.  
**kPDB\_PulseOutChannel2Mask** PDB pulse out trigger channel number 2 mask.  
**kPDB\_PulseOutChannel3Mask** PDB pulse out trigger channel number 3 mask.

## 22.6 Function Documentation

### 22.6.1 void PDB\_Init ( **PDB\_Type** \* *base*, **const pdb\_config\_t** \* *config* )

This function initializes the PDB module. The operations included are as follows.

- Enable the clock for PDB instance.
- Configure the PDB module.
- Enable the PDB module.

## Parameters

|               |                                                             |
|---------------|-------------------------------------------------------------|
| <i>base</i>   | PDB peripheral base address.                                |
| <i>config</i> | Pointer to the configuration structure. See "pdb_config_t". |

**22.6.2 void PDB\_Deinit ( PDB\_Type \* *base* )**

## Parameters

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

**22.6.3 void PDB\_GetDefaultConfig ( pdb\_config\_t \* *config* )**

This function initializes the user configuration structure to a default value. The default values are as follows.

```
* config->loadValueMode = kPDB_LoadValueImmediately;
* config->prescalerDivider = kPDB_PrescalerDivider1;
* config->dividerMultiplicationFactor = kPDB_DividerMultiplicationFactor1
 ;
* config->triggerInputSource = kPDB_TriggerSoftware;
* config->enableContinuousMode = false;
*
```

## Parameters

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

**22.6.4 static void PDB\_Enable ( PDB\_Type \* *base*, bool *enable* ) [inline], [static]**

## Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PDB peripheral base address. |
| <i>enable</i> | Enable the module or not.    |

**22.6.5 static void PDB\_DoSoftwareTrigger ( PDB\_Type \* *base* ) [inline], [static]**

Parameters

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

### 22.6.6 static void PDB\_DoLoadValues ( PDB\_Type \* *base* ) [inline], [static]

This function loads the counter values from the internal buffer. See "pdb\_load\_value\_mode\_t" about PDB's load mode.

Parameters

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

### 22.6.7 static void PDB\_EnableDMA ( PDB\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PDB peripheral base address. |
| <i>enable</i> | Enable the feature or not.   |

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

Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>base</i> | PDB peripheral base address.                            |
| <i>mask</i> | Mask value for interrupts. See "_pdb_interrupt_enable". |

### 22.6.9 static void PDB\_DisableInterrupts ( PDB\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>base</i> | PDB peripheral base address.                            |
| <i>mask</i> | Mask value for interrupts. See "_pdb_interrupt_enable". |

### 22.6.10 static uint32\_t PDB\_GetStatusFlags ( PDB\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

Mask value for asserted flags. See "\_pdb\_status\_flags".

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

Parameters

|             |                                               |
|-------------|-----------------------------------------------|
| <i>base</i> | PDB peripheral base address.                  |
| <i>mask</i> | Mask value of flags. See "_pdb_status_flags". |

### 22.6.12 static void PDB\_SetModulusValue ( PDB\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Parameters

|              |                                                     |
|--------------|-----------------------------------------------------|
| <i>base</i>  | PDB peripheral base address.                        |
| <i>value</i> | Setting value for the modulus. 16-bit is available. |

### 22.6.13 static uint32\_t PDB\_GetCounterValue ( PDB\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

PDB counter's current value.

#### 22.6.14 static void PDB\_SetCounterDelayValue ( PDB\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Parameters

|              |                                                                 |
|--------------|-----------------------------------------------------------------|
| <i>base</i>  | PDB peripheral base address.                                    |
| <i>value</i> | Setting value for PDB counter delay event. 16-bit is available. |

#### 22.6.15 static void PDB\_SetADCPreTriggerConfig ( PDB\_Type \* *base*, pdb\_adc\_trigger\_channel\_t *channel*, pdb\_adc\_pretrigger\_config\_t \* *config* ) [inline], [static]

Parameters

|                |                                                                            |
|----------------|----------------------------------------------------------------------------|
| <i>base</i>    | PDB peripheral base address.                                               |
| <i>channel</i> | Channel index for ADC instance.                                            |
| <i>config</i>  | Pointer to the configuration structure. See "pdb_adc_pretrigger_config_t". |

#### 22.6.16 static void PDB\_SetADCPreTriggerDelayValue ( PDB\_Type \* *base*, pdb\_adc\_trigger\_channel\_t *channel*, pdb\_adc\_pretrigger\_t *pretriggerNumber*, uint32\_t *value* ) [inline], [static]

This function sets the value for ADC pre-trigger delay event. It specifies the delay value for the channel's corresponding pre-trigger. The pre-trigger asserts when the PDB counter is equal to the set value.

Parameters

|                          |                                                                     |
|--------------------------|---------------------------------------------------------------------|
| <i>base</i>              | PDB peripheral base address.                                        |
| <i>channel</i>           | Channel index for ADC instance.                                     |
| <i>pretrigger-Number</i> | Channel group index for ADC instance.                               |
| <i>value</i>             | Setting value for ADC pre-trigger delay event. 16-bit is available. |

**22.6.17 static uint32\_t PDB\_GetADCPreTriggerStatusFlags ( PDB\_Type \* *base*, pdb\_adc\_trigger\_channel\_t *channel* ) [inline], [static]**

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | PDB peripheral base address.    |
| <i>channel</i> | Channel index for ADC instance. |

Returns

Mask value for asserted flags. See "\_pdb\_adc\_pretrigger\_flags".

**22.6.18 static void PDB\_ClearADCPreTriggerStatusFlags ( PDB\_Type \* *base*, pdb\_adc\_trigger\_channel\_t *channel*, uint32\_t *mask* ) [inline], [static]**

Parameters

|                |                                                        |
|----------------|--------------------------------------------------------|
| <i>base</i>    | PDB peripheral base address.                           |
| <i>channel</i> | Channel index for ADC instance.                        |
| <i>mask</i>    | Mask value for flags. See "_pdb_adc_pretrigger_flags". |

**22.6.19 void PDB\_SetDACTriggerConfig ( PDB\_Type \* *base*, pdb\_dac\_trigger\_channel\_t *channel*, pdb\_dac\_trigger\_config\_t \* *config* )**

Parameters

|                |                                                                         |
|----------------|-------------------------------------------------------------------------|
| <i>base</i>    | PDB peripheral base address.                                            |
| <i>channel</i> | Channel index for DAC instance.                                         |
| <i>config</i>  | Pointer to the configuration structure. See "pdb_dac_trigger_config_t". |

**22.6.20 static void PDB\_SetDACTriggerIntervalValue ( PDB\_Type \* *base*, pdb\_dac\_trigger\_channel\_t *channel*, uint32\_t *value* ) [inline], [static]**

This function sets the value for DAC interval event. DAC interval trigger triggers the DAC module to update the buffer when the DAC interval counter is equal to the set value.

Parameters

|                |                                           |
|----------------|-------------------------------------------|
| <i>base</i>    | PDB peripheral base address.              |
| <i>channel</i> | Channel index for DAC instance.           |
| <i>value</i>   | Setting value for the DAC interval event. |

**22.6.21 static void PDB\_EnablePulseOutTrigger ( PDB\_Type \* *base*, pdb\_pulse\_out\_channel\_mask\_t *channelMask*, bool *enable* ) [inline], [static]**

Parameters

|                    |                                                            |
|--------------------|------------------------------------------------------------|
| <i>base</i>        | PDB peripheral base address.                               |
| <i>channelMask</i> | Channel mask value for multiple pulse out trigger channel. |
| <i>enable</i>      | Whether the feature is enabled or not.                     |

**22.6.22 static void PDB\_SetPulseOutTriggerDelayValue ( PDB\_Type \* *base*, pdb\_pulse\_out\_trigger\_channel\_t *channel*, uint32\_t *value1*, uint32\_t *value2* ) [inline], [static]**

This function is used to set event values for the pulse output trigger. These pulse output trigger delay values specify the delay for the PDB Pulse-out. Pulse-out goes high when the PDB counter is equal to the pulse output high value (value1). Pulse-out goes low when the PDB counter is equal to the pulse output low value (value2).

## Parameters

|                |                                              |
|----------------|----------------------------------------------|
| <i>base</i>    | PDB peripheral base address.                 |
| <i>channel</i> | Channel index for pulse out trigger channel. |
| <i>value1</i>  | Setting value for pulse out high.            |
| <i>value2</i>  | Setting value for pulse out low.             |

# Chapter 23

## PIT: Periodic Interrupt Timer

### 23.1 Overview

The MCUXpresso SDK provides a driver for the Periodic Interrupt Timer (PIT) of MCUXpresso SDK devices.

### 23.2 Function groups

The PIT driver supports operating the module as a time counter.

#### 23.2.1 Initialization and deinitialization

The function [PIT\\_Init\(\)](#) initializes the PIT with specified configurations. The function [PIT\\_GetDefaultConfig\(\)](#) gets the default configurations. The initialization function configures the PIT operation in debug mode.

The function [PIT\\_SetTimerChainMode\(\)](#) configures the chain mode operation of each PIT channel.

The function [PIT\\_Deinit\(\)](#) disables the PIT timers and disables the module clock.

#### 23.2.2 Timer period Operations

The function [PITR\\_SetTimerPeriod\(\)](#) sets the timer period in units of count. Timers begin counting down from the value set by this function until it reaches 0.

The function [PIT\\_GetCurrentTimerCount\(\)](#) reads the current timer counting value. This function returns the real-time timer counting value, in a range from 0 to a timer period.

The timer period operation functions takes the count value in ticks. Users can call the utility macros provided in `fsl_common.h` to convert to microseconds or milliseconds.

#### 23.2.3 Start and Stop timer operations

The function [PIT\\_StartTimer\(\)](#) starts the timer counting. After calling this function, the timer loads the period value set earlier via the [PIT\\_SetPeriod\(\)](#) function and starts counting down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

The function [PIT\\_StopTimer\(\)](#) stops the timer counting.

### 23.2.4 Status

Provides functions to get and clear the PIT status.

### 23.2.5 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

## 23.3 Typical use case

### 23.3.1 PIT tick example

Updates the PIT period and toggles an LED periodically. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/pit

## Data Structures

- struct [pit\\_config\\_t](#)  
*PIT configuration structure. [More...](#)*

## Enumerations

- enum [pit\\_chnl\\_t](#) {
   
   kPIT\_Chnl\_0 = 0U,
   
   kPIT\_Chnl\_1,
   
   kPIT\_Chnl\_2,
   
   kPIT\_Chnl\_3
 }
   
*List of PIT channels.*
- enum [pit\\_interrupt\\_enable\\_t](#) { kPIT\_TimerInterruptEnable = PIT\_TCTRL\_TIE\_MASK }
   
*List of PIT interrupts.*
- enum [pit\\_status\\_flags\\_t](#) { kPIT\_TimerFlag = PIT\_TFLG\_TIF\_MASK }
   
*List of PIT status flags.*

## Driver version

- #define [FSL\\_PIT\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 4))
   
*PIT Driver Version 2.0.4.*

## Initialization and deinitialization

- void [PIT\\_Init](#) (PIT\_Type \*base, const [pit\\_config\\_t](#) \*config)
   
*Ungates the PIT clock, enables the PIT module, and configures the peripheral for basic operations.*
- void [PIT\\_Deinit](#) (PIT\_Type \*base)
   
*Gates the PIT clock and disables the PIT module.*
- static void [PIT\\_GetDefaultConfig](#) ([pit\\_config\\_t](#) \*config)
   
*Fills in the PIT configuration structure with the default settings.*
- static void [PIT\\_SetTimerChainMode](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel, bool enable)
   
*Enables or disables chaining a timer with the previous timer.*

## Interrupt Interface

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

## Status Interface

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

## Read and Write the timer period

- static void [PIT\\_SetTimerPeriod](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel, uint32\_t count)  
*Sets the timer period in units of count.*
- static uint32\_t [PIT\\_GetCurrentTimerCount](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel)  
*Reads the current timer counting value.*

## Timer Start and Stop

- static void [PIT\\_StartTimer](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel)  
*Starts the timer counting.*
- static void [PIT\\_StopTimer](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel)  
*Stops the timer counting.*

## 23.4 Data Structure Documentation

### 23.4.1 struct pit\_config\_t

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

The configuration structure can be made constant so it resides in flash.

## Data Fields

- bool [enableRunInDebug](#)  
*true: Timers run in debug mode; false: Timers stop in debug mode*

## 23.5 Enumeration Type Documentation

### 23.5.1 enum pit\_chnl\_t

Note

Actual number of available channels is SoC dependent

Enumerator

- kPIT\_Chnl\_0*** PIT channel number 0.
- kPIT\_Chnl\_1*** PIT channel number 1.
- kPIT\_Chnl\_2*** PIT channel number 2.
- kPIT\_Chnl\_3*** PIT channel number 3.

### 23.5.2 enum pit\_interrupt\_enable\_t

Enumerator

- kPIT\_TimerInterruptEnable*** Timer interrupt enable.

### 23.5.3 enum pit\_status\_flags\_t

Enumerator

- kPIT\_TimerFlag*** Timer flag.

## 23.6 Function Documentation

### 23.6.1 void PIT\_Init ( **PIT\_Type** \* *base*, **const pit\_config\_t** \* *config* )

Note

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

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | PIT peripheral base address                |
| <i>config</i> | Pointer to the user's PIT config structure |

### 23.6.2 void PIT\_Deinit ( **PIT\_Type** \* *base* )

Parameters

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

### 23.6.3 static void PIT\_GetDefaultConfig ( pit\_config\_t \* *config* ) [inline], [static]

The default values are as follows.

```
* config->enableRunInDebug = false;
*
```

Parameters

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

### 23.6.4 static void PIT\_SetTimerChainMode ( PIT\_Type \* *base*, pit\_chnl\_t *channel*, bool *enable* ) [inline], [static]

When a timer has a chain mode enabled, it only counts after the previous timer has expired. If the timer n-1 has counted down to 0, counter n decrements the value by one. Each timer is 32-bits, which allows the developers to chain timers together and form a longer timer (64-bits and larger). The first timer (timer 0) can't be chained to any other timer.

Parameters

|                |                                                                                                                                |
|----------------|--------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | PIT peripheral base address                                                                                                    |
| <i>channel</i> | Timer channel number which is chained with the previous timer                                                                  |
| <i>enable</i>  | Enable or disable chain. true: Current timer is chained with the previous timer. false: Timer doesn't chain with other timers. |

### 23.6.5 static void PIT\_EnableInterrupts ( PIT\_Type \* *base*, pit\_chnl\_t *channel*, uint32\_t *mask* ) [inline], [static]

Parameters

|                |                                                                                                                     |
|----------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | PIT peripheral base address                                                                                         |
| <i>channel</i> | Timer channel number                                                                                                |
| <i>mask</i>    | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">pit_interrupt_enable_t</a> |

### 23.6.6 static void PIT\_DisableInterrupts ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel*, **uint32\_t** *mask* ) [inline], [static]

Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | PIT peripheral base address                                                                                          |
| <i>channel</i> | Timer channel number                                                                                                 |
| <i>mask</i>    | The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">pit_interrupt_enable_t</a> |

### 23.6.7 static **uint32\_t** PIT\_GetEnabledInterrupts ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel* ) [inline], [static]

Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number        |

Returns

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

### 23.6.8 static **uint32\_t** PIT\_GetStatusFlags ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel* ) [inline], [static]

Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number        |

Returns

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

### 23.6.9 static void PIT\_ClearStatusFlags ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel*, **uint32\_t** *mask* ) [inline], [static]

Parameters

|                |                                                                                                                  |
|----------------|------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | PIT peripheral base address                                                                                      |
| <i>channel</i> | Timer channel number                                                                                             |
| <i>mask</i>    | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">pit_status_flags_t</a> |

### 23.6.10 static void PIT\_SetTimerPeriod ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel*, **uint32\_t** *count* ) [inline], [static]

Timers begin counting from the value set by this function until it reaches 0, then it generates an interrupt and load this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

Note

Users can call the utility macros provided in `fsl_common.h` to convert to ticks.

Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number        |

|              |                                |
|--------------|--------------------------------|
| <i>count</i> | Timer period in units of ticks |
|--------------|--------------------------------|

### 23.6.11 static uint32\_t PIT\_GetCurrentTimerCount ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel* ) [inline], [static]

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note

Users can call the utility macros provided in fsl\_common.h to convert ticks to usec or msec.

Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number        |

Returns

Current timer counting value in ticks

### 23.6.12 static void PIT\_StartTimer ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel* ) [inline], [static]

After calling this function, timers load period value, count down to 0 and then load the respective start value again. Each time a timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number.       |

### 23.6.13 static void PIT\_StopTimer ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel* ) [inline], [static]

This function stops every timer counting. Timers reload their periods respectively after the next time they call the PIT\_DRV\_StartTimer.

## Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number.       |

# Chapter 24

## PMC: Power Management Controller

### 24.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Power Management Controller (PMC) module of MCUXpresso SDK devices. The PMC module contains internal voltage regulator, power on reset, low-voltage detect system, and high-voltage detect system.

### Data Structures

- struct [pmc\\_low\\_volt\\_detect\\_config\\_t](#)  
*Low-voltage Detect Configuration Structure. [More...](#)*
- struct [pmc\\_low\\_volt\\_warning\\_config\\_t](#)  
*Low-voltage Warning Configuration Structure. [More...](#)*
- struct [pmc\\_bandgap\\_buffer\\_config\\_t](#)  
*Bandgap Buffer configuration. [More...](#)*

### Enumerations

- enum [pmc\\_low\\_volt\\_detect\\_volt\\_select\\_t](#) {  
  kPMC\_LowVoltDetectLowTrip = 0U,  
  kPMC\_LowVoltDetectHighTrip = 1U }  
*Low-voltage Detect Voltage Select.*
- enum [pmc\\_low\\_volt\\_warning\\_volt\\_select\\_t](#) {  
  kPMC\_LowVoltWarningLowTrip = 0U,  
  kPMC\_LowVoltWarningMid1Trip = 1U,  
  kPMC\_LowVoltWarningMid2Trip = 2U,  
  kPMC\_LowVoltWarningHighTrip = 3U }  
*Low-voltage Warning Voltage Select.*

### Driver version

- #define [FSL\\_PMC\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 3))  
*PMC driver version.*

### Power Management Controller Control APIs

- void [PMC\\_ConfigureLowVoltDetect](#) (PMC\_Type \*base, const [pmc\\_low\\_volt\\_detect\\_config\\_t](#) \*config)  
*Configures the low-voltage detect setting.*
- static bool [PMC\\_GetLowVoltDetectFlag](#) (PMC\_Type \*base)  
*Gets the Low-voltage Detect Flag status.*
- static void [PMC\\_ClearLowVoltDetectFlag](#) (PMC\_Type \*base)  
*Acknowledges clearing the Low-voltage Detect flag.*

- void **PMC\_ConfigureLowVoltWarning** (PMC\_Type \*base, const **pmc\_low\_volt\_warning\_config\_t** \*config)  
*Configures the low-voltage warning setting.*
- static bool **PMC\_GetLowVoltWarningFlag** (PMC\_Type \*base)  
*Gets the Low-voltage Warning Flag status.*
- static void **PMC\_ClearLowVoltWarningFlag** (PMC\_Type \*base)  
*Acknowledges the Low-voltage Warning flag.*
- void **PMC\_ConfigureBandgapBuffer** (PMC\_Type \*base, const **pmc\_bandgap\_buffer\_config\_t** \*config)  
*Configures the PMC bandgap.*
- static bool **PMC\_GetPeriphIOIsolationFlag** (PMC\_Type \*base)  
*Gets the acknowledge Peripherals and I/O pads isolation flag.*
- static void **PMC\_ClearPeriphIOIsolationFlag** (PMC\_Type \*base)  
*Acknowledges the isolation flag to Peripherals and I/O pads.*
- static bool **PMC\_IsRegulatorInRunRegulation** (PMC\_Type \*base)  
*Gets the regulator regulation status.*

## 24.2 Data Structure Documentation

### 24.2.1 struct pmc\_low\_volt\_detect\_config\_t

#### Data Fields

- bool **enableInt**  
*Enable interrupt when Low-voltage detect.*
- bool **enableReset**  
*Enable system reset when Low-voltage detect.*
- **pmc\_low\_volt\_detect\_volt\_select\_t** **voltSelect**  
*Low-voltage detect trip point voltage selection.*

### 24.2.2 struct pmc\_low\_volt\_warning\_config\_t

#### Data Fields

- bool **enableInt**  
*Enable interrupt when low-voltage warning.*
- **pmc\_low\_volt\_warning\_volt\_select\_t** **voltSelect**  
*Low-voltage warning trip point voltage selection.*

### 24.2.3 struct pmc\_bandgap\_buffer\_config\_t

#### Data Fields

- bool **enable**  
*Enable bandgap buffer.*
- bool **enableInLowPowerMode**

*Enable bandgap buffer in low-power mode.*

#### Field Documentation

- (1) `bool pmc_bandgap_buffer_config_t::enable`
- (2) `bool pmc_bandgap_buffer_config_t::enableInLowPowerMode`

### 24.3 Macro Definition Documentation

#### 24.3.1 `#define FSL_PMC_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))`

Version 2.0.3.

### 24.4 Enumeration Type Documentation

#### 24.4.1 `enum pmc_low_volt_detect_volt_select_t`

Enumerator

- kPMC\_LowVoltDetectLowTrip* Low-trip point selected (VLVD = VLVDL )
- kPMC\_LowVoltDetectHighTrip* High-trip point selected (VLVD = VLVDH )

#### 24.4.2 `enum pmc_low_volt_warning_volt_select_t`

Enumerator

- kPMC\_LowVoltWarningLowTrip* Low-trip point selected (VLVW = VLVW1)
- kPMC\_LowVoltWarningMid1Trip* Mid 1 trip point selected (VLVW = VLVW2)
- kPMC\_LowVoltWarningMid2Trip* Mid 2 trip point selected (VLVW = VLVW3)
- kPMC\_LowVoltWarningHighTrip* High-trip point selected (VLVW = VLVW4)

### 24.5 Function Documentation

#### 24.5.1 `void PMC_ConfigureLowVoltDetect ( PMC_Type * base, const pmc_low_volt_detect_config_t * config )`

This function configures the low-voltage detect setting, including the trip point voltage setting, enables or disables the interrupt, enables or disables the system reset.

Parameters

|               |                                             |
|---------------|---------------------------------------------|
| <i>base</i>   | PMC peripheral base address.                |
| <i>config</i> | Low-voltage detect configuration structure. |

#### 24.5.2 static bool PMC\_GetLowVoltDetectFlag ( **PMC\_Type** \* *base* ) [inline], [static]

This function reads the current LVDF status. If it returns 1, a low-voltage event is detected.

Parameters

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

Returns

Current low-voltage detect flag

- true: Low-voltage detected
- false: Low-voltage not detected

#### 24.5.3 static void PMC\_ClearLowVoltDetectFlag ( **PMC\_Type** \* *base* ) [inline], [static]

This function acknowledges the low-voltage detection errors (write 1 to clear LVDF).

Parameters

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

#### 24.5.4 void PMC\_ConfigureLowVoltWarning ( **PMC\_Type** \* *base*, const **pmc\_low\_volt\_warning\_config\_t** \* *config* )

This function configures the low-voltage warning setting, including the trip point voltage setting and enabling or disabling the interrupt.

Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>base</i>   | PMC peripheral base address.                 |
| <i>config</i> | Low-voltage warning configuration structure. |

#### 24.5.5 static bool PMC\_GetLowVoltWarningFlag ( **PMC\_Type** \* *base* ) [inline], [static]

This function polls the current LVWF status. When 1 is returned, it indicates a low-voltage warning event. LVWF is set when V Supply transitions below the trip point or after reset and V Supply is already below the V LVW.

Parameters

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

Returns

Current LVWF status

- true: Low-voltage Warning Flag is set.
- false: the Low-voltage Warning does not happen.

#### 24.5.6 static void PMC\_ClearLowVoltWarningFlag ( **PMC\_Type** \* *base* ) [inline], [static]

This function acknowledges the low voltage warning errors (write 1 to clear LVWF).

Parameters

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

#### 24.5.7 void PMC\_ConfigureBandgapBuffer ( **PMC\_Type** \* *base*, const **pmc\_bandgap\_buffer\_config\_t** \* *config* )

This function configures the PMC bandgap, including the drive select and behavior in low-power mode.

Parameters

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

#### 24.5.8 static bool PMC\_GetPeriphIOIsolationFlag ( **PMC\_Type** \* *base* ) [**inline**], [**static**]

This function reads the Acknowledge Isolation setting that indicates whether certain peripherals and the I/O pads are in a latched state as a result of having been in the VLLS mode.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | PMC peripheral base address.           |
| <i>base</i> | Base address for current PMC instance. |

Returns

ACK isolation 0 - Peripherals and I/O pads are in a normal run state. 1 - Certain peripherals and I/O pads are in an isolated and latched state.

#### 24.5.9 static void PMC\_ClearPeriphIOIsolationFlag ( **PMC\_Type** \* *base* ) [**inline**], [**static**]

This function clears the ACK Isolation flag. Writing one to this setting when it is set releases the I/O pads and certain peripherals to their normal run mode state.

Parameters

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

#### 24.5.10 static bool PMC\_IsRegulatorInRunRegulation ( **PMC\_Type** \* *base* ) [**inline**], [**static**]

This function returns the regulator to run a regulation status. It provides the current status of the internal voltage regulator.

## Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | PMC peripheral base address.           |
| <i>base</i> | Base address for current PMC instance. |

## Returns

Regulation status 0 - Regulator is in a stop regulation or in transition to/from the regulation. 1 - Regulator is in a run regulation.

# Chapter 25

## PORT: Port Control and Interrupts

### 25.1 Overview

The MCUXpresso SDK provides a driver for the Port Control and Interrupts (PORT) module of MCUXpresso SDK devices.

### Data Structures

- struct `port_digital_filter_config_t`  
*PORT digital filter feature configuration definition.* [More...](#)
- struct `port_pin_config_t`  
*PORT pin configuration structure.* [More...](#)

### Enumerations

- enum `_port_pull` {  
  `kPORT_PullDisable` = 0U,  
  `kPORT_PullDown` = 2U,  
  `kPORT_PullUp` = 3U }  
*Internal resistor pull feature selection.*
- enum `_port_slew_rate` {  
  `kPORT_FastSlewRate` = 0U,  
  `kPORT_SlowSlewRate` = 1U }  
*Slew rate selection.*
- enum `_port_open_drain_enable` {  
  `kPORT_OpenDrainDisable` = 0U,  
  `kPORT_OpenDrainEnable` = 1U }  
*Open Drain feature enable/disable.*
- enum `_port_passive_filter_enable` {  
  `kPORT_PassiveFilterDisable` = 0U,  
  `kPORT_PassiveFilterEnable` = 1U }  
*Passive filter feature enable/disable.*
- enum `_port_drive_strength` {  
  `kPORT_LowDriveStrength` = 0U,  
  `kPORT_HighDriveStrength` = 1U }  
*Configures the drive strength.*
- enum `_port_lock_register` {  
  `kPORT_UnlockRegister` = 0U,  
  `kPORT_LockRegister` = 1U }  
*Unlock/lock the pin control register field[15:0].*
- enum `port_mux_t` {

```
kPORT_PinDisabledOrAnalog = 0U,
kPORT_MuxAsGpio = 1U,
kPORT_MuxAlt2 = 2U,
kPORT_MuxAlt3 = 3U,
kPORT_MuxAlt4 = 4U,
kPORT_MuxAlt5 = 5U,
kPORT_MuxAlt6 = 6U,
kPORT_MuxAlt7 = 7U,
kPORT_MuxAlt8 = 8U,
kPORT_MuxAlt9 = 9U,
kPORT_MuxAlt10 = 10U,
kPORT_MuxAlt11 = 11U,
kPORT_MuxAlt12 = 12U,
kPORT_MuxAlt13 = 13U,
kPORT_MuxAlt14 = 14U,
kPORT_MuxAlt15 = 15U }
```

*Pin mux selection.*

- enum `port_interrupt_t` {
 

```
kPORT_InterruptOrDMADisabled = 0x0U,
kPORT_DMARisingEdge = 0x1U,
kPORT_DMAFallingEdge = 0x2U,
kPORT_DMAEitherEdge = 0x3U,
kPORT_FlagRisingEdge = 0x05U,
kPORT_FlagFallingEdge = 0x06U,
kPORT_FlagEitherEdge = 0x07U,
kPORT_InterruptLogicZero = 0x8U,
kPORT_InterruptRisingEdge = 0x9U,
kPORT_InterruptFallingEdge = 0xAU,
kPORT_InterruptEitherEdge = 0xBU,
kPORT_InterruptLogicOne = 0xCU,
kPORT_ActiveHighTriggerOutputEnable = 0xDU,
kPORT_ActiveLowTriggerOutputEnable = 0xEU }
```

*Configures the interrupt generation condition.*

- enum `port_digital_filter_clock_source_t` {
 

```
kPORT_BusClock = 0U,
kPORT_LpoClock = 1U }
```

*Digital filter clock source selection.*

## Driver version

- #define `FSL_PORT_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 0)`)  
*PORT driver version.*

## Configuration

- static void `PORT_SetPinConfig` (`PORT_Type` \*base, `uint32_t` pin, const `port_pin_config_t` \*config)

- static void [PORT\\_SetMultiplePinsConfig](#) (PORT\_Type \*base, uint32\_t mask, const [port\\_pin\\_config\\_t](#) \*config)
 

*Sets the port PCR register.*
- static void [PORT\\_SetPinMux](#) (PORT\_Type \*base, uint32\_t pin, [port\\_mux\\_t](#) mux)
 

*Sets the port PCR register for multiple pins.*
- static void [PORT\\_EnablePinsDigitalFilter](#) (PORT\_Type \*base, uint32\_t mask, bool enable)
 

*Configures the pin muxing.*
- static void [PORT\\_SetDigitalFilterConfig](#) (PORT\_Type \*base, const [port\\_digital\\_filter\\_config\\_t](#) \*config)
 

*Enables the digital filter in one port, each bit of the 32-bit register represents one pin.*
- static uint32\_t [PORT\\_GetPinsInterruptFlags](#) (PORT\_Type \*base)
 

*Reads the whole port status flag.*
- static void [PORT\\_ClearPinsInterruptFlags](#) (PORT\_Type \*base, uint32\_t mask)
 

*Clears the multiple pin interrupt status flag.*

## Interrupt

- static void [PORT\\_SetPinInterruptConfig](#) (PORT\_Type \*base, uint32\_t pin, [port\\_interrupt\\_t](#) config)
 

*Configures the port pin interrupt/DMA request.*
- static void [PORT\\_SetPinDriveStrength](#) (PORT\_Type \*base, uint32\_t pin, uint8\_t strength)
 

*Configures the port pin drive strength.*
- static uint32\_t [PORT\\_GetPinsInterruptFlags](#) (PORT\_Type \*base)
 

*Reads the whole port status flag.*
- static void [PORT\\_ClearPinsInterruptFlags](#) (PORT\_Type \*base, uint32\_t mask)
 

*Clears the multiple pin interrupt status flag.*

## 25.2 Data Structure Documentation

### 25.2.1 struct port\_digital\_filter\_config\_t

#### Data Fields

- uint32\_t [digitalFilterWidth](#)

*Set digital filter width.*
- [port\\_digital\\_filter\\_clock\\_source\\_t](#) [clockSource](#)

*Set digital filter clockSource.*

### 25.2.2 struct port\_pin\_config\_t

#### Data Fields

- uint16\_t [pullSelect](#): 2
 

*No-pull/pull-down/pull-up select.*
- uint16\_t [slewRate](#): 1
 

*Fast/slow slew rate Configure.*
- uint16\_t [passiveFilterEnable](#): 1
 

*Passive filter enable/disable.*
- uint16\_t [openDrainEnable](#): 1
 

*Open drain enable/disable.*
- uint16\_t [driveStrength](#): 1
 

*Fast/slow drive strength configure.*

- `uint16_t mux`: 3  
*Pin mux Configure.*
- `uint16_t lockRegister`: 1  
*Lock/unlock the PCR field[15:0].*

## 25.3 Macro Definition Documentation

### 25.3.1 #define FSL\_PORT\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 0))

## 25.4 Enumeration Type Documentation

### 25.4.1 enum \_port\_pull

Enumerator

***kPORT\_PullDisable*** Internal pull-up/down resistor is disabled.

***kPORT\_PullDown*** Internal pull-down resistor is enabled.

***kPORT\_PullUp*** Internal pull-up resistor is enabled.

### 25.4.2 enum \_port\_slew\_rate

Enumerator

***kPORT\_FastSlewRate*** Fast slew rate is configured.

***kPORT\_SlowSlewRate*** Slow slew rate is configured.

### 25.4.3 enum \_port\_open\_drain\_enable

Enumerator

***kPORT\_OpenDrainDisable*** Open drain output is disabled.

***kPORT\_OpenDrainEnable*** Open drain output is enabled.

### 25.4.4 enum \_port\_passive\_filter\_enable

Enumerator

***kPORT\_PassiveFilterDisable*** Passive input filter is disabled.

***kPORT\_PassiveFilterEnable*** Passive input filter is enabled.

## 25.4.5 enum \_port\_drive\_strength

Enumerator

***kPORT\_LowDriveStrength*** Low-drive strength is configured.

***kPORT\_HighDriveStrength*** High-drive strength is configured.

## 25.4.6 enum \_port\_lock\_register

Enumerator

***kPORT\_UnlockRegister*** Pin Control Register fields [15:0] are not locked.

***kPORT\_LockRegister*** Pin Control Register fields [15:0] are locked.

## 25.4.7 enum port\_mux\_t

Enumerator

***kPORT\_PinDisabledOrAnalog*** Corresponding pin is disabled, but is used as an analog pin.

***kPORT\_MuxAsGpio*** Corresponding pin is configured as GPIO.

***kPORT\_MuxAlt2*** Chip-specific.

***kPORT\_MuxAlt3*** Chip-specific.

***kPORT\_MuxAlt4*** Chip-specific.

***kPORT\_MuxAlt5*** Chip-specific.

***kPORT\_MuxAlt6*** Chip-specific.

***kPORT\_MuxAlt7*** Chip-specific.

***kPORT\_MuxAlt8*** Chip-specific.

***kPORT\_MuxAlt9*** Chip-specific.

***kPORT\_MuxAlt10*** Chip-specific.

***kPORT\_MuxAlt11*** Chip-specific.

***kPORT\_MuxAlt12*** Chip-specific.

***kPORT\_MuxAlt13*** Chip-specific.

***kPORT\_MuxAlt14*** Chip-specific.

***kPORT\_MuxAlt15*** Chip-specific.

## 25.4.8 enum port\_interrupt\_t

Enumerator

***kPORT\_InterruptOrDMA.Disabled*** Interrupt/DMA request is disabled.

***kPORT\_DMARisingEdge*** DMA request on rising edge.

***kPORT\_DMAPFallingEdge*** DMA request on falling edge.

*kPORT\_DMAEitherEdge* DMA request on either edge.  
*kPORT\_FlagRisingEdge* Flag sets on rising edge.  
*kPORT\_FlagFallingEdge* Flag sets on falling edge.  
*kPORT\_FlagEitherEdge* Flag sets on either edge.  
*kPORT\_InterruptLogicZero* Interrupt when logic zero.  
*kPORT\_InterruptRisingEdge* Interrupt on rising edge.  
*kPORT\_InterruptFallingEdge* Interrupt on falling edge.  
*kPORT\_InterruptEitherEdge* Interrupt on either edge.  
*kPORT\_InterruptLogicOne* Interrupt when logic one.  
*kPORT\_ActiveHighTriggerOutputEnable* Enable active high-trigger output.  
*kPORT\_ActiveLowTriggerOutputEnable* Enable active low-trigger output.

## 25.4.9 enum port\_digital\_filter\_clock\_source\_t

Enumerator

*kPORT\_BusClock* Digital filters are clocked by the bus clock.  
*kPORT\_LpoClock* Digital filters are clocked by the 1 kHz LPO clock.

## 25.5 Function Documentation

### 25.5.1 static void PORT\_SetPinConfig ( *PORT\_Type* \* *base*, *uint32\_t* *pin*, const *port\_pin\_config\_t* \* *config* ) [inline], [static]

This is an example to define an input pin or output pin PCR configuration.

```
* // Define a digital input pin PCR configuration
* port_pin_config_t config = {
* kPORT_PullUp,
* kPORT_FastSlewRate,
* kPORT_PassiveFilterDisable,
* kPORT_OpenDrainDisable,
* kPORT_LowDriveStrength,
* kPORT_MuxAsGpio,
* kPORT_UnLockRegister,
* };
*
```

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | PORT peripheral base pointer. |
|-------------|-------------------------------|

|               |                                            |
|---------------|--------------------------------------------|
| <i>pin</i>    | PORT pin number.                           |
| <i>config</i> | PORT PCR register configuration structure. |

### 25.5.2 static void PORT\_SetMultiplePinsConfig ( PORT\_Type \* *base*, uint32\_t *mask*, const port\_pin\_config\_t \* *config* ) [inline], [static]

This is an example to define input pins or output pins PCR configuration.

```
* Define a digital input pin PCR configuration
* port_pin_config_t config = {
* kPORT_PullUp,
* kPORT_PullEnable,
* kPORT_FastSlewRate,
* kPORT_PassiveFilterDisable,
* kPORT_OpenDrainDisable,
* kPORT_LowDriveStrength,
* kPORT_MuxAsGpio,
* kPORT_UnlockRegister,
* };
*
```

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | PORT peripheral base pointer.              |
| <i>mask</i>   | PORT pin number macro.                     |
| <i>config</i> | PORT PCR register configuration structure. |

### 25.5.3 static void PORT\_SetPinMux ( PORT\_Type \* *base*, uint32\_t *pin*, port\_mux\_t *mux* ) [inline], [static]

Parameters

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | PORT peripheral base pointer.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <i>pin</i>  | PORT pin number.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <i>mux</i>  | pin muxing slot selection. <ul style="list-style-type: none"> <li>• <a href="#">kPORT_PinDisabledOrAnalog</a>: Pin disabled or work in analog function.</li> <li>• <a href="#">kPORT_MuxAsGpio</a> : Set as GPIO.</li> <li>• <a href="#">kPORT_MuxAlt2</a> : chip-specific.</li> <li>• <a href="#">kPORT_MuxAlt3</a> : chip-specific.</li> <li>• <a href="#">kPORT_MuxAlt4</a> : chip-specific.</li> <li>• <a href="#">kPORT_MuxAlt5</a> : chip-specific.</li> <li>• <a href="#">kPORT_MuxAlt6</a> : chip-specific.</li> <li>• <a href="#">kPORT_MuxAlt7</a> : chip-specific.</li> </ul> |

## Note

: This function is NOT recommended to use together with the PORT\_SetPinsConfig, because the PORT\_SetPinsConfig need to configure the pin mux anyway (Otherwise the pin mux is reset to zero : kPORT\_PinDisabledOrAnalog). This function is recommended to use to reset the pin mux

#### **25.5.4 static void PORT\_EnablePinsDigitalFilter ( PORT\_Type \* *base*, uint32\_t *mask*, bool *enable* ) [inline], [static]**

## Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | PORT peripheral base pointer.      |
| <i>mask</i>   | PORT pin number macro.             |
| <i>enable</i> | PORT digital filter configuration. |

#### **25.5.5 static void PORT\_SetDigitalFilterConfig ( PORT\_Type \* *base*, const port\_digital\_filter\_config\_t \* *config* ) [inline], [static]**

## Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>base</i>   | PORT peripheral base pointer.                |
| <i>config</i> | PORT digital filter configuration structure. |

### 25.5.6 static void PORT\_SetPinInterruptConfig ( **PORT\_Type** \* *base*, **uint32\_t** *pin*, **port\_interrupt\_t** *config* ) [inline], [static]

Parameters

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | PORT peripheral base pointer.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <i>pin</i>    | PORT pin number.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <i>config</i> | <p>PORT pin interrupt configuration.</p> <ul style="list-style-type: none"> <li>• <a href="#">kPORT_InterruptOrDMADisabled</a>: Interrupt/DMA request disabled.</li> <li>• <a href="#">kPORT_DMARisingEdge</a> : DMA request on rising edge(if the DMA requests exit).</li> <li>• <a href="#">kPORT_DMAPFallingEdge</a>: DMA request on falling edge(if the DMA requests exit).</li> <li>• <a href="#">kPORT_DMAEitherEdge</a> : DMA request on either edge(if the DMA requests exit).</li> <li>• <a href="#">kPORT_FlagRisingEdge</a> : Flag sets on rising edge(if the Flag states exit).</li> <li>• <a href="#">kPORT_FlagFallingEdge</a> : Flag sets on falling edge(if the Flag states exit).</li> <li>• <a href="#">kPORT_FlagEitherEdge</a> : Flag sets on either edge(if the Flag states exit).</li> <li>• <a href="#">kPORT_InterruptLogicZero</a> : Interrupt when logic zero.</li> <li>• <a href="#">kPORT_InterruptRisingEdge</a> : Interrupt on rising edge.</li> <li>• <a href="#">kPORT_InterruptFallingEdge</a>: Interrupt on falling edge.</li> <li>• <a href="#">kPORT_InterruptEitherEdge</a> : Interrupt on either edge.</li> <li>• <a href="#">kPORT_InterruptLogicOne</a> : Interrupt when logic one.</li> <li>• <a href="#">kPORT_ActiveHighTriggerOutputEnable</a> : Enable active high-trigger output (if the trigger states exit).</li> <li>• <a href="#">kPORT_ActiveLowTriggerOutputEnable</a> : Enable active low-trigger output (if the trigger states exit).</li> </ul> |

### 25.5.7 static void PORT\_SetPinDriveStrength ( **PORT\_Type** \* *base*, **uint32\_t** *pin*, **uint8\_t** *strength* ) [inline], [static]

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | PORT peripheral base pointer. |
| <i>pin</i>  | PORT pin number.              |

|                 |                                                                                                                                                                                                                                                          |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>strength</i> | PORt pin drive strength <ul style="list-style-type: none"> <li>• <a href="#">kPORT_LowDriveStrength</a> = 0U - Low-drive strength is configured.</li> <li>• <a href="#">kPORT_HighDriveStrength</a> = 1U - High-drive strength is configured.</li> </ul> |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 25.5.8 static uint32\_t PORT\_GetPinsInterruptFlags ( **PORT\_Type** \* *base* ) [**inline**], [**static**]

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | PORt peripheral base pointer. |
|-------------|-------------------------------|

Returns

Current port interrupt status flags, for example, 0x00010001 means the pin 0 and 16 have the interrupt.

### 25.5.9 static void PORT\_ClearPinsInterruptFlags ( **PORT\_Type** \* *base*, uint32\_t *mask* ) [**inline**], [**static**]

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | PORt peripheral base pointer. |
| <i>mask</i> | PORt pin number macro.        |

# Chapter 26

## RCM: Reset Control Module Driver

### 26.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Reset Control Module (RCM) module of MCUXpresso SDK devices.

### Data Structures

- struct `rcm_reset_pin_filter_config_t`  
*Reset pin filter configuration. [More...](#)*

### Enumerations

- enum `rcm_reset_source_t`{  
    `kRCM_SourceWakeup` = RCM\_SRS0\_WAKEUP\_MASK,  
    `kRCM_SourceLvd` = RCM\_SRS0\_LVD\_MASK,  
    `kRCM_SourceLoc` = RCM\_SRS0\_LOC\_MASK,  
    `kRCM_SourceLol` = RCM\_SRS0\_LOL\_MASK,  
    `kRCM_SourceWdog` = RCM\_SRS0\_WDOG\_MASK,  
    `kRCM_SourcePin` = RCM\_SRS0\_PIN\_MASK,  
    `kRCM_SourcePor` = RCM\_SRS0\_POR\_MASK,  
    `kRCM_SourceJtag` = RCM\_SRS1\_JTAG\_MASK << 8U,  
    `kRCM_SourceLockup` = RCM\_SRS1\_LOCKUP\_MASK << 8U,  
    `kRCM_SourceSw` = RCM\_SRS1\_SW\_MASK << 8U,  
    `kRCM_SourceMdmap` = RCM\_SRS1\_MDM\_AP\_MASK << 8U,  
    `kRCM_SourceEzpt` = RCM\_SRS1\_EZPT\_MASK << 8U,  
    `kRCM_SourceSackerr` = RCM\_SRS1\_SACKERR\_MASK << 8U }  
    *System Reset Source Name definitions.*

- enum `rcm_run_wait_filter_mode_t`{  
    `kRCM_FilterDisable` = 0U,  
    `kRCM_FilterBusClock` = 1U,  
    `kRCM_FilterLpoClock` = 2U }  
    *Reset pin filter select in Run and Wait modes.*

### Driver version

- #define `FSL_RCM_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 4))  
*RCM driver version 2.0.4.*

### Reset Control Module APIs

- static uint32\_t `RCM_GetPreviousResetSources` (RCM\_Type \*base)

- static uint32\_t [RCM\\_GetStickyResetSources](#) (RCM\_Type \*base)
 

*Gets the reset source status which caused a previous reset.*
- static void [RCM\\_ClearStickyResetSources](#) (RCM\_Type \*base, uint32\_t sourceMasks)
 

*Gets the sticky reset source status.*
- void [RCM\\_ConfigureResetPinFilter](#) (RCM\_Type \*base, const [rcm\\_reset\\_pin\\_filter\\_config\\_t](#) \*config)
 

*Clears the sticky reset source status.*
- static bool [RCM\\_GetEasyPortModePinStatus](#) (RCM\_Type \*base)
 

*Configures the reset pin filter.*
- Gets the EZP\_MS\_B pin assert status.

## 26.2 Data Structure Documentation

### 26.2.1 struct rcm\_reset\_pin\_filter\_config\_t

#### Data Fields

- bool [enableFilterInStop](#)

*Reset pin filter select in stop mode.*
- [rcm\\_run\\_wait\\_filter\\_mode\\_t](#) [filterInRunWait](#)

*Reset pin filter in run/wait mode.*
- uint8\_t [busClockFilterCount](#)

*Reset pin bus clock filter width.*

#### Field Documentation

- (1) [bool rcm\\_reset\\_pin\\_filter\\_config\\_t::enableFilterInStop](#)
- (2) [rcm\\_run\\_wait\\_filter\\_mode\\_t rcm\\_reset\\_pin\\_filter\\_config\\_t::filterInRunWait](#)
- (3) [uint8\\_t rcm\\_reset\\_pin\\_filter\\_config\\_t::busClockFilterCount](#)

## 26.3 Macro Definition Documentation

### 26.3.1 #define FSL\_RCM\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 4))

## 26.4 Enumeration Type Documentation

### 26.4.1 enum rcm\_reset\_source\_t

Enumerator

- kRCM\_SourceWakeup* Low-leakage wakeup reset.
- kRCM\_SourceLvd* Low-voltage detect reset.
- kRCM\_SourceLoc* Loss of clock reset.
- kRCM\_SourceLol* Loss of lock reset.
- kRCM\_SourceWdog* Watchdog reset.
- kRCM\_SourcePin* External pin reset.
- kRCM\_SourcePor* Power on reset.

*kRCM\_SourceJtag* JTAG generated reset.  
*kRCM\_SourceLockup* Core lock up reset.  
*kRCM\_SourceSw* Software reset.  
*kRCM\_SourceMdmap* MDM-AP system reset.  
*kRCM\_SourceEzpt* EzPort reset.  
*kRCM\_SourceSackerr* Parameter could get all reset flags.

## 26.4.2 enum rcm\_run\_wait\_filter\_mode\_t

Enumerator

*kRCM\_FilterDisable* All filtering disabled.  
*kRCM\_FilterBusClock* Bus clock filter enabled.  
*kRCM\_FilterLpoClock* LPO clock filter enabled.

## 26.5 Function Documentation

### 26.5.1 static uint32\_t RCM\_GetPreviousResetSources ( RCM\_Type \* *base* ) [inline], [static]

This function gets the current reset source status. Use source masks defined in the rcm\_reset\_source\_t to get the desired source status.

This is an example.

```
* uint32_t resetStatus;
*
* To get all reset source statuses.
* resetStatus = RCM_GetPreviousResetSources(RCM) & kRCM_SourceAll;
*
* To test whether the MCU is reset using Watchdog.
* resetStatus = RCM_GetPreviousResetSources(RCM) &
 kRCM_SourceWdog;
*
* To test multiple reset sources.
* resetStatus = RCM_GetPreviousResetSources(RCM) & (
 kRCM_SourceWdog | kRCM_SourcePin);
*
```

Parameters

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

Returns

All reset source status bit map.

### 26.5.2 static uint32\_t RCM\_GetStickyResetSources ( RCM\_Type \* *base* ) [inline], [static]

This function gets the current reset source status that has not been cleared by software for a specific source.  
This is an example.

```
* uint32_t resetStatus;
*
* To get all reset source statuses.
* resetStatus = RCM_GetStickyResetSources(RCM) & kRCM_SourceAll;
*
* To test whether the MCU is reset using Watchdog.
* resetStatus = RCM_GetStickyResetSources(RCM) &
 kRCM_SourceWdog;
*
* To test multiple reset sources.
* resetStatus = RCM_GetStickyResetSources(RCM) & (
 kRCM_SourceWdog | kRCM_SourcePin);
*
```

Parameters

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

Returns

All reset source status bit map.

### 26.5.3 static void RCM\_ClearStickyResetSources ( RCM\_Type \* *base*, uint32\_t *sourceMasks* ) [inline], [static]

This function clears the sticky system reset flags indicated by source masks.

This is an example.

```
* Clears multiple reset sources.
* RCM_ClearStickyResetSources(kRCM_SourceWdog |
 kRCM_SourcePin);
*
```

Parameters

|                    |                              |
|--------------------|------------------------------|
| <i>base</i>        | RCM peripheral base address. |
| <i>sourceMasks</i> | reset source status bit map  |

#### 26.5.4 void RCM\_ConfigureResetPinFilter ( RCM\_Type \* *base*, const rcm\_reset\_pin\_filter\_config\_t \* *config* )

This function sets the reset pin filter including the filter source, filter width, and so on.

Parameters

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

#### 26.5.5 static bool RCM\_GetEasyPortModePinStatus ( RCM\_Type \* *base* ) [inline], [static]

This function gets the easy port mode status (EZP\_MS\_B) pin assert status.

Parameters

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

Returns

status true - asserted, false - reasserted

# Chapter 27

## RNGA: Random Number Generator Accelerator Driver

### 27.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Random Number Generator Accelerator (RNGA) block of MCUXpresso SDK devices.

### 27.2 RNGA Initialization

1. To initialize the RNGA module, call the [RNGA\\_Init\(\)](#) function. This function automatically enables the RNGA module and its clock.
2. After calling the [RNGA\\_Init\(\)](#) function, the RNGA is enabled and the counter starts working.
3. To disable the RNGA module, call the [RNGA\\_Deinit\(\)](#) function.

### 27.3 Get random data from RNGA

1. [RNGA\\_GetRandomData\(\)](#) function gets random data from the RNGA module.

### 27.4 RNGA Set/Get Working Mode

The RNGA works either in sleep mode or normal mode

1. [RNGA\\_SetMode\(\)](#) function sets the RNGA mode.
2. [RNGA\\_GetMode\(\)](#) function gets the RNGA working mode.

### 27.5 Seed RNGA

1. [RNGA\\_Seed\(\)](#) function inputs an entropy value that the RNGA can use to seed the pseudo random algorithm.

This example code shows how to initialize and get random data from the RNGA driver:

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/rnga

#### Note

It is important to note that there is no known cryptographic proof showing this is a secure method for generating random data. In fact, there may be an attack against this random number generator if its output is used directly in a cryptographic application. The attack is based on the linearity of the internal shift registers. Therefore, it is highly recommended that the random data produced by this module be used as an entropy source to provide an input seed to a NIST-approved pseudo-random-number generator based on DES or SHA-1 and defined in NIST FIPS PUB 186-2 Appendix 3 and NIST FIPS PUB SP 800-90. The requirement is needed to maximize the entropy of this input seed. To do this, when data is extracted from RNGA as quickly as the hardware allows, there are one to two bits of added entropy per 32-bit word. Any single bit of that word contains that entropy.

Therefore, when used as an entropy source, a random number should be generated for each bit of entropy required and the least significant bit (any bit would be equivalent) of each word retained. The remainder of each random number should then be discarded. Used this way, even with full knowledge of the internal state of RNGA and all prior random numbers, an attacker is not able to predict the values of the extracted bits. Other sources of entropy can be used along with RNGA to generate the seed to the pseudorandom algorithm. The more random sources combined to create the seed, the better. The following is a list of sources that can be easily combined with the output of this module.

- Current time using highest precision possible
- Real-time system inputs that can be characterized as "random"
- Other entropy supplied directly by the user

## Enumerations

- enum `rnga_mode_t` {
   
    `kRNGA_ModeNormal` = 0U,
   
    `kRNGA_ModeSleep` = 1U
 }
- RNGA working mode.*

## Functions

- void `RNGA_Init` (RNG\_Type \*base)  
*Initializes the RNGA.*
- void `RNGA_Deinit` (RNG\_Type \*base)  
*Shuts down the RNGA.*
- status\_t `RNGA_GetRandomData` (RNG\_Type \*base, void \*data, size\_t data\_size)  
*Gets random data.*
- void `RNGA_Seed` (RNG\_Type \*base, uint32\_t seed)  
*Feeds the RNGA module.*
- void `RNGA_SetMode` (RNG\_Type \*base, `rnga_mode_t` mode)  
*Sets the RNGA in normal mode or sleep mode.*
- `rnga_mode_t RNGA_GetMode` (RNG\_Type \*base)  
*Gets the RNGA working mode.*

## Driver version

- #define `FSL_RNGA_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 2))  
*RNGA driver version 2.0.2.*

## 27.6 Macro Definition Documentation

### 27.6.1 #define FSL\_RNGA\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))

## 27.7 Enumeration Type Documentation

### 27.7.1 enum rnga\_mode\_t

Enumerator

**kRNGA\_ModeNormal** Normal Mode. The ring-oscillator clocks are active; RNGA generates entropy (randomness) from the clocks and stores it in shift registers.

**kRNGA\_ModeSleep** Sleep Mode. The ring-oscillator clocks are inactive; RNGA does not generate entropy.

## 27.8 Function Documentation

### 27.8.1 void RNGA\_Init ( RNG\_Type \* *base* )

This function initializes the RNGA. When called, the RNGA entropy generation starts immediately.

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | RNGA base address |
|-------------|-------------------|

### 27.8.2 void RNGA\_Deinit ( RNG\_Type \* *base* )

This function shuts down the RNGA.

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | RNGA base address |
|-------------|-------------------|

### 27.8.3 status\_t RNGA\_GetRandomData ( RNG\_Type \* *base*, void \* *data*, size\_t *data\_size* )

This function gets random data from the RNGA.

Parameters

|                  |                                                    |
|------------------|----------------------------------------------------|
| <i>base</i>      | RNGA 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

RNGA status

#### 27.8.4 void RNGA\_Seed ( RNG\_Type \* *base*, uint32\_t *seed* )

This function inputs an entropy value that the RNGA uses to seed its pseudo-random algorithm.

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | RNGA base address |
| <i>seed</i> | input seed value  |

### 27.8.5 void RNGA\_SetMode ( RNG\_Type \* *base*, rnga\_mode\_t *mode* )

This function sets the RNGA in sleep mode or normal mode.

Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | RNGA base address         |
| <i>mode</i> | normal mode or sleep mode |

### 27.8.6 rnga\_mode\_t RNGA\_GetMode ( RNG\_Type \* *base* )

This function gets the RNGA working mode.

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | RNGA base address |
|-------------|-------------------|

Returns

normal mode or sleep mode

# Chapter 28

## RTC: Real Time Clock

### 28.1 Overview

The MCUXpresso SDK provides a driver for the Real Time Clock (RTC) of MCUXpresso SDK devices.

### 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. Users pass 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. Users pass 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.

## 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 RTC Oscillator

Some SoC's allow control of the RTC oscillator through the RTC module.

The function `RTC_SetOscCapLoad()` allows the user to modify the capacitor load configuration of the RTC oscillator.

## 28.2.8 Monotonic Counter

Some SoC's have a 64-bit Monotonic counter available in the RTC module.

The function `RTC_SetMonotonicCounter()` writes a 64-bit to the counter.

The function `RTC_GetMonotonicCounter()` reads the monotonic counter and returns the 64-bit counter value to the user.

The function `RTC_IncrementMonotonicCounter()` increments the Monotonic Counter by one.

## 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`

## Data Structures

- struct `rtc_datetime_t`  
*Structure is used to hold the date and time. [More...](#)*
- struct `rtc_config_t`  
*RTC config structure. [More...](#)*

## Enumerations

- enum `rtc_interrupt_enable_t` {
   
  `kRTC_TimeInvalidInterruptEnable` = (1U << 0U),
   
  `kRTC_TimeOverflowInterruptEnable` = (1U << 1U),
   
  `kRTC_AlarmInterruptEnable` = (1U << 2U),
 }

- ```

kRTC_SecondsInterruptEnable = (1U << 3U) }

List of RTC interrupts.
• enum rtc_status_flags_t {
    kRTC_TimeInvalidFlag = (1U << 0U),
    kRTC_TimeOverflowFlag = (1U << 1U),
    kRTC_AlarmFlag = (1U << 2U) }

List of RTC flags.
• enum rtc_osc_cap_load_t {
    kRTC_Capacitor_2p = RTC_CR_SC2P_MASK,
    kRTC_Capacitor_4p = RTC_CR_SC4P_MASK,
    kRTC_Capacitor_8p = RTC_CR_SC8P_MASK,
    kRTC_Capacitor_16p = RTC_CR_SC16P_MASK }

List of RTC Oscillator capacitor load settings.

```

Functions

- static void **RTC_SetClockSource** (RTC_Type *base)
Set RTC clock source.
- static void **RTC_SetOscCapLoad** (RTC_Type *base, uint32_t capLoad)
This function sets the specified capacitor configuration for the RTC oscillator.
- static void **RTC_Reset** (RTC_Type *base)
Performs a software reset on the RTC module.
- static void **RTC_EnableWakeUpPin** (RTC_Type *base, bool enable)
Enables or disables the RTC Wakeup Pin Operation.

Driver version

- #define **FSL_RTC_DRIVER_VERSION** (MAKE_VERSION(2, 2, 1))
Version 2.2.1.

Initialization and deinitialization

- void **RTC_Init** (RTC_Type *base, const **rtc_config_t** *config)
Ungates the RTC clock and configures the peripheral for basic operation.
- static void **RTC_Deinit** (RTC_Type *base)
Stops the timer and gate the RTC clock.
- void **RTC_GetDefaultConfig** (**rtc_config_t** *config)
Fills in the RTC config struct with the default settings.

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

- void [RTC_EnableInterrupts](#) (RTC_Type *base, uint32_t mask)
Enables the selected RTC interrupts.
- void [RTC_DisableInterrupts](#) (RTC_Type *base, uint32_t mask)
Disables the selected RTC interrupts.
- uint32_t [RTC_GetEnabledInterrupts](#) (RTC_Type *base)
Gets the enabled RTC interrupts.

Status Interface

- uint32_t [RTC_GetStatusFlags](#) (RTC_Type *base)
Gets the RTC status flags.
- 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.

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.

Field Documentation

- (1) `uint16_t rtc_datetime_t::year`
- (2) `uint8_t rtc_datetime_t::month`
- (3) `uint8_t rtc_datetime_t::day`

- (4) `uint8_t rtc_datetime_t::hour`
- (5) `uint8_t rtc_datetime_t::minute`
- (6) `uint8_t rtc_datetime_t::second`

28.4.2 struct rtc_config_t

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the `RTC_GetDefaultConfig()` function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

Data Fields

- bool `wakeupSelect`
true: Wakeup pin outputs the 32 KHz clock; false: Wakeup pin used to wakeup the chip
- bool `updateMode`
true: Registers can be written even when locked under certain conditions, false: No writes allowed when registers are locked
- bool `supervisorAccess`
true: Non-supervisor accesses are allowed; false: Non-supervisor accesses are not supported
- uint32_t `compensationInterval`
Compensation interval that is written to the CIR field in RTC TCR Register.
- uint32_t `compensationTime`
Compensation time that is written to the TCR field in RTC TCR Register.

28.5 Enumeration Type Documentation

28.5.1 enum rtc_interrupt_enable_t

Enumerator

- `kRTC_TimeInvalidInterruptEnable`** Time invalid interrupt.
- `kRTC_TimeOverflowInterruptEnable`** Time overflow interrupt.
- `kRTC_AlarmInterruptEnable`** Alarm interrupt.
- `kRTC_SecondsInterruptEnable`** Seconds interrupt.

28.5.2 enum rtc_status_flags_t

Enumerator

- `kRTC_TimeInvalidFlag`** Time invalid flag.
- `kRTC_TimeOverflowFlag`** Time overflow flag.
- `kRTC_AlarmFlag`** Alarm flag.

28.5.3 enum rtc_osc_cap_load_t

Enumerator

<i>kRTC_Capacitor_2p</i>	2 pF capacitor load
<i>kRTC_Capacitor_4p</i>	4 pF capacitor load
<i>kRTC_Capacitor_8p</i>	8 pF capacitor load
<i>kRTC_Capacitor_16p</i>	16 pF capacitor load

28.6 Function Documentation

28.6.1 void RTC_Init (RTC_Type * *base*, const rtc_config_t * *config*)

This function issues a software reset if the timer invalid flag is set.

Note

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

Parameters

<i>base</i>	RTC peripheral base address
<i>config</i>	Pointer to the user's RTC configuration structure.

28.6.2 static void RTC_Deinit (RTC_Type * *base*) [inline], [static]

Parameters

<i>base</i>	RTC peripheral base address
-------------	-----------------------------

28.6.3 void RTC_GetDefaultConfig (rtc_config_t * *config*)

The default values are as follows.

```
* config->wakeupSelect = false;
* config->updateMode = false;
* config->supervisorAccess = false;
* config->compensationInterval = 0;
* config->compensationTime = 0;
*
```

Parameters

<i>config</i>	Pointer to the user's RTC configuration structure.
---------------	--

28.6.4 **status_t RTC_SetDatetime (RTC_Type * *base*, const rtc_datetime_t * *datetime*)**

The RTC counter must be stopped prior to calling this function because writes to the RTC seconds register fail if the RTC counter is running.

Parameters

<i>base</i>	RTC peripheral base address
<i>datetime</i>	Pointer to the structure where the date and time details are stored.

Returns

kStatus_Success: Success in setting the time and starting the RTC
kStatus_InvalidArgument: Error because the datetime format is incorrect

28.6.5 **void RTC_GetDatetime (RTC_Type * *base*, rtc_datetime_t * *datetime*)**

Parameters

<i>base</i>	RTC peripheral base address
<i>datetime</i>	Pointer to the structure where the date and time details are stored.

28.6.6 **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 the 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

28.6.7 void RTC_GetAlarm (RTC_Type * *base*, rtc_datetime_t * *datetime*)

Parameters

<i>base</i>	RTC peripheral base address
<i>datetime</i>	Pointer to the structure where the alarm date and time details are stored.

28.6.8 void RTC_EnableInterrupts (RTC_Type * *base*, uint32_t *mask*)

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 rtc_interrupt_enable_t

28.6.9 void RTC_DisableInterrupts (RTC_Type * *base*, uint32_t *mask*)

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 rtc_interrupt_enable_t

28.6.10 uint32_t RTC_GetEnabledInterrupts (RTC_Type * *base*)

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](#)

28.6.11 **uint32_t RTC_GetStatusFlags (RTC_Type * *base*)**

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](#)

28.6.12 **void RTC_ClearStatusFlags (RTC_Type * *base*, uint32_t *mask*)**

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 rtc_status_flags_t

28.6.13 **static void RTC_SetClockSource (RTC_Type * *base*) [inline], [static]**

Parameters

<i>base</i>	RTC peripheral base address
-------------	-----------------------------

Note

After setting this bit, wait the oscillator startup time before enabling the time counter to allow the 32.768 kHz clock time to stabilize.

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_SetOscCapLoad (RTC_Type * *base*, uint32_t *capLoad*) [inline], [static]

Parameters

<i>base</i>	RTC peripheral base address
<i>capLoad</i>	Oscillator loads to enable. This is a logical OR of members of the enumeration rtc_osc_cap_load_t

28.6.17 static void RTC_Reset (RTC_Type * *base*) [inline], [static]

This resets all RTC registers except for the SWR bit and the RTC_WAR and RTC_RAR registers. The SWR bit is cleared by software explicitly clearing it.

Parameters

<i>base</i>	RTC peripheral base address
-------------	-----------------------------

28.6.18 static void RTC_EnableWakeUpPin (RTC_Type * *base*, bool *enable*) [inline], [static]

This function enable or disable RTC Wakeup Pin. The wakeup pin is optional and not available on all devices.

Parameters

<i>base</i>	RTC_Type base pointer.
<i>enable</i>	true to enable, false to disable.

Chapter 29

SAI: Serial Audio Interface

29.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Serial Audio Interface (SAI) module of MCUXpresso SDK devices.

SAI driver includes functional APIs and transactional APIs.

Functional APIs target low-level APIs. Functional APIs can be used for SAI initialization, configuration and operation, and for optimization and customization purposes. Using the functional API requires the knowledge of the SAI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SAI functional operation groups provide the functional API set.

Transactional APIs 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 `sai_handle_t` as the first parameter. Initialize the handle by calling the [SAI_TransferTxCreateHandle\(\)](#) or [SAI_TransferRxCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [SAI_TransferSendNonBlocking\(\)](#) and [SAI_TransferReceiveNonBlocking\(\)](#) set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_SAI_TxIdle` and `kStatus_SAI_RxIdle` status.

29.2 Typical configurations

Bit width configuration

SAI driver support 8/16/24/32bits stereo/mono raw audio data transfer. SAI EDMA driver support 8/16/32bits stereo/mono raw audio data transfer, since the EDMA doesn't support 24bit data width, so application should pre-convert the 24bit data to 32bit. SAI DMA driver support 8/16/32bits stereo/mono raw audio data transfer, since the EDMA doesn't support 24bit data width, so application should pre-convert the 24bit data to 32bit. SAI SDMA driver support 8/16/24/32bits stereo/mono raw audio data transfer.

Frame configuration

SAI driver support I2S, DSP, Left justified, Right justified, TDM mode. Application can call the api directly: `SAI_GetClassicI2SConfig` `SAI_GetLeftJustifiedConfig` `SAI_GetRightJustifiedConfig` `SAI_GetTDMConfig` `SAI_GetDSPConfig`

29.3 Typical use case

29.3.1 SAI Send/receive using an interrupt method

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

29.3.2 SAI Send/receive using a DMA method

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

Modules

- [SAI Driver](#)
- [SAI EDMA Driver](#)

29.4 SAI Driver

29.4.1 Overview

Data Structures

- struct `sai_config_t`
SAI user configuration structure. [More...](#)
- struct `sai_transfer_format_t`
sai transfer format [More...](#)
- struct `sai_master_clock_t`
master clock configurations [More...](#)
- struct `sai_fifo_t`
sai fifo configurations [More...](#)
- struct `sai_bit_clock_t`
sai bit clock configurations [More...](#)
- struct `sai_frame_sync_t`
sai frame sync configurations [More...](#)
- struct `sai_serial_data_t`
sai serial data configurations [More...](#)
- struct `sai_transceiver_t`
sai transceiver configurations [More...](#)
- struct `sai_transfer_t`
SAI transfer structure. [More...](#)
- struct `sai_handle_t`
SAI handle structure. [More...](#)

Macros

- #define `SAI_XFER_QUEUE_SIZE` (4U)
SAI transfer queue size, user can refine it according to use case.
- #define `FSL_SAI_HAS_FIFO_EXTEND_FEATURE` 1
sai fifo feature

Typedefs

- typedef void(* `sai_transfer_callback_t`)(I2S_Type *base, sai_handle_t *handle, `status_t` status, void *userData)
SAI transfer callback prototype.

Enumerations

- enum {

kStatus_SAI_TxBusy = MAKE_STATUS(kStatusGroup_SAI, 0),

kStatus_SAI_RxBusy = MAKE_STATUS(kStatusGroup_SAI, 1),

kStatus_SAI_TxError = MAKE_STATUS(kStatusGroup_SAI, 2),

kStatus_SAI_RxError = MAKE_STATUS(kStatusGroup_SAI, 3),

kStatus_SAI_QueueFull = MAKE_STATUS(kStatusGroup_SAI, 4),

kStatus_SAI_TxIdle = MAKE_STATUS(kStatusGroup_SAI, 5),

kStatus_SAI_RxIdle = MAKE_STATUS(kStatusGroup_SAI, 6) }

_sai_status_t, SAI return status.
- enum {

kSAI_Channel0Mask = 1 << 0U,

kSAI_Channel1Mask = 1 << 1U,

kSAI_Channel2Mask = 1 << 2U,

kSAI_Channel3Mask = 1 << 3U,

kSAI_Channel4Mask = 1 << 4U,

kSAI_Channel5Mask = 1 << 5U,

kSAI_Channel6Mask = 1 << 6U,

kSAI_Channel7Mask = 1 << 7U } }

_sai_channel_mask,.sai channel mask value, actual channel numbers is depend soc specific
- enum **sai_protocol_t** {

kSAI_BusLeftJustified = 0x0U,

kSAI_BusRightJustified,

kSAI_BusI2S,

kSAI_BusPCMA,

kSAI_BusPCMB }
- Define the SAI bus type.*
- enum **sai_master_slave_t** {

kSAI_Master = 0x0U,

kSAI_Slave = 0x1U,

kSAI_Bclk_Master_FrameSync_Slave = 0x2U,

kSAI_Bclk_Slave_FrameSync_Master = 0x3U } }

Master or slave mode.
- enum **sai_mono_stereo_t** {

kSAI_Stereo = 0x0U,

kSAI_MonoRight,

kSAI_MonoLeft }
- Mono or stereo audio format.*
- enum **sai_data_order_t** {

kSAI_DataLSB = 0x0U,

kSAI_DataMSB }
- SAI data order, MSB or LSB.*
- enum **sai_clock_polarity_t** {

- kSAI_PolarityActiveHigh = 0x0U,
 kSAI_PolarityActiveLow = 0x1U,
 kSAI_SampleOnFallingEdge = 0x0U,
 kSAI_SampleOnRisingEdge = 0x1U }
- SAI clock polarity, active high or low.*
- enum `sai_sync_mode_t` {
 kSAI_ModeAsync = 0x0U,
 kSAI_ModeSync }
- Synchronous or asynchronous mode.*
- enum `sai_mclk_source_t` {
 kSAI_MclkSourceSysclk = 0x0U,
 kSAI_MclkSourceSelect1,
 kSAI_MclkSourceSelect2,
 kSAI_MclkSourceSelect3 }
- Mater clock source.*
- enum `sai_bclk_source_t` {
 kSAI_BclkSourceBusclk = 0x0U,
 kSAI_BclkSourceMclkOption1 = 0x1U,
 kSAI_BclkSourceMclkOption2 = 0x2U,
 kSAI_BclkSourceMclkOption3 = 0x3U,
 kSAI_BclkSourceMclkDiv = 0x1U,
 kSAI_BclkSourceOtherSai0 = 0x2U,
 kSAI_BclkSourceOtherSai1 = 0x3U }
- Bit clock source.*
- enum {
 kSAI_WordStartInterruptEnable,
 kSAI_SyncErrorInterruptEnable = I2S_TCSR_SEIE_MASK,
 kSAI_FIFOWarningInterruptEnable = I2S_TCSR_FWIE_MASK,
 kSAI_FIFOErrorInterruptEnable = I2S_TCSR_FEIE_MASK,
 kSAI_FIFORequestInterruptEnable = I2S_TCSR_FRIE_MASK }
- _sai_interrupt_enable_t, The SAI interrupt enable flag*
- enum {
 kSAI_FIFOWarningDMAEnable = I2S_TCSR_FWDE_MASK,
 kSAI_FIFORequestDMAEnable = I2S_TCSR_FRDE_MASK }
- _sai_dma_enable_t, The DMA request sources*
- enum {
 kSAI_WordStartFlag = I2S_TCSR_WSF_MASK,
 kSAI_SyncErrorFlag = I2S_TCSR_SEF_MASK,
 kSAI_FIFOErrorFlag = I2S_TCSR_FEF_MASK,
 kSAI_FIFORequestFlag = I2S_TCSR_FRF_MASK,
 kSAI_FIFOWarningFlag = I2S_TCSR_FWF_MASK }
- _sai_flags, The SAI status flag*
- enum `sai_reset_type_t` {
 kSAI_ResetTypeSoftware = I2S_TCSR_SR_MASK,
 kSAI_ResetTypeFIFO = I2S_TCSR_FR_MASK,
 kSAI_ResetAll = I2S_TCSR_SR_MASK | I2S_TCSR_FR_MASK }

The reset type.

- enum `sai_fifo_packing_t` {

 `kSAI_FifoPackingDisabled` = 0x0U,

 `kSAI_FifoPacking8bit` = 0x2U,

 `kSAI_FifoPacking16bit` = 0x3U }

The SAI packing mode The mode includes 8 bit and 16 bit packing.

- enum `sai_sample_rate_t` {

 `kSAI_SampleRate8KHz` = 8000U,

 `kSAI_SampleRate11025Hz` = 11025U,

 `kSAI_SampleRate12KHz` = 12000U,

 `kSAI_SampleRate16KHz` = 16000U,

 `kSAI_SampleRate22050Hz` = 22050U,

 `kSAI_SampleRate24KHz` = 24000U,

 `kSAI_SampleRate32KHz` = 32000U,

 `kSAI_SampleRate44100Hz` = 44100U,

 `kSAI_SampleRate48KHz` = 48000U,

 `kSAI_SampleRate96KHz` = 96000U,

 `kSAI_SampleRate192KHz` = 192000U,

 `kSAI_SampleRate384KHz` = 384000U }

Audio sample rate.

- enum `sai_word_width_t` {

 `kSAI_WordWidth8bits` = 8U,

 `kSAI_WordWidth16bits` = 16U,

 `kSAI_WordWidth24bits` = 24U,

 `kSAI_WordWidth32bits` = 32U }

Audio word width.

- enum `sai_transceiver_type_t` {

 `kSAI_Transmitter` = 0U,

 `kSAI_Receiver` = 1U }
- sai transceiver type*
- enum `sai_frame_sync_len_t` {

 `kSAI_FrameSyncLenOneBitClk` = 0U,

 `kSAI_FrameSyncLenPerWordWidth` = 1U }

sai frame sync len

Driver version

- #define `FSL_SAI_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 6)`)
- Version 2.3.6.*

Initialization and deinitialization

- void `SAL_TxInit` (I2S_Type *base, const `sai_config_t` *config)

 Initializes the SAI Tx peripheral.
- void `SAL_RxInit` (I2S_Type *base, const `sai_config_t` *config)

- **`void SAI_TxGetDefaultConfig (sai_config_t *config)`**

Initializes the SAI Rx peripheral.

Sets the SAI Tx configuration structure to default values.
- **`void SAI_RxGetDefaultConfig (sai_config_t *config)`**

Sets the SAI Rx configuration structure to default values.
- **`void SAI_Init (I2S_Type *base)`**

Initializes the SAI peripheral.
- **`void SAI_Deinit (I2S_Type *base)`**

De-initializes the SAI peripheral.
- **`void SAI_TxReset (I2S_Type *base)`**

Resets the SAI Tx.
- **`void SAI_RxReset (I2S_Type *base)`**

Resets the SAI Rx.
- **`void SAI_TxEnable (I2S_Type *base, bool enable)`**

Enables/disables the SAI Tx.
- **`void SAI_RxEnable (I2S_Type *base, bool enable)`**

Enables/disables the SAI Rx.
- **`static void SAI_TxSetBitClockDirection (I2S_Type *base, sai_master_slave_t masterSlave)`**

Set Rx bit clock direction.
- **`static void SAI_RxSetBitClockDirection (I2S_Type *base, sai_master_slave_t masterSlave)`**

Set Rx bit clock direction.
- **`static void SAI_RxSetFrameSyncDirection (I2S_Type *base, sai_master_slave_t masterSlave)`**

Set Rx frame sync direction.
- **`static void SAI_TxSetFrameSyncDirection (I2S_Type *base, sai_master_slave_t masterSlave)`**

Set Tx frame sync direction.
- **`void SAI_TxSetBitClockRate (I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)`**

Transmitter bit clock rate configurations.
- **`void SAI_RxSetBitClockRate (I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)`**

Receiver bit clock rate configurations.
- **`void SAI_TxSetBitclockConfig (I2S_Type *base, sai_master_slave_t masterSlave, sai_bit_clock_t *config)`**

Transmitter Bit clock configurations.
- **`void SAI_RxSetBitclockConfig (I2S_Type *base, sai_master_slave_t masterSlave, sai_bit_clock_t *config)`**

Receiver Bit clock configurations.
- **`void SAI_SetMasterClockConfig (I2S_Type *base, sai_master_clock_t *config)`**

Master clock configurations.
- **`void SAI_TxSetFifoConfig (I2S_Type *base, sai_fifo_t *config)`**

SAI transmitter fifo configurations.
- **`void SAI_RxSetFifoConfig (I2S_Type *base, sai_fifo_t *config)`**

SAI receiver fifo configurations.
- **`void SAI_TxSetFrameSyncConfig (I2S_Type *base, sai_master_slave_t masterSlave, sai_frame_sync_t *config)`**

SAI transmitter Frame sync configurations.
- **`void SAI_RxSetFrameSyncConfig (I2S_Type *base, sai_master_slave_t masterSlave, sai_frame_sync_t *config)`**

SAI receiver Frame sync configurations.
- **`void SAI_TxSetSerialDataConfig (I2S_Type *base, sai_serial_data_t *config)`**

- *SAI transmitter Serial data configurations.*
- void **`SAI_RxSetSerialDataConfig`** (I2S_Type *base, `sai_serial_data_t` *config)
SAI receiver Serial data configurations.
- void **`SAI_TxSetConfig`** (I2S_Type *base, `sai_transceiver_t` *config)
SAI transmitter configurations.
- void **`SAI_RxSetConfig`** (I2S_Type *base, `sai_transceiver_t` *config)
SAI receiver configurations.
- void **`SAI_GetClassicI2SConfig`** (`sai_transceiver_t` *config, `sai_word_width_t` bitWidth, `sai_mono_stereo_t` mode, `uint32_t` saiChannelMask)
Get classic I2S mode configurations.
- void **`SAI_GetLeftJustifiedConfig`** (`sai_transceiver_t` *config, `sai_word_width_t` bitWidth, `sai_mono_stereo_t` mode, `uint32_t` saiChannelMask)
Get left justified mode configurations.
- void **`SAI_GetRightJustifiedConfig`** (`sai_transceiver_t` *config, `sai_word_width_t` bitWidth, `sai_mono_stereo_t` mode, `uint32_t` saiChannelMask)
Get right justified mode configurations.
- void **`SAI_GetTDMConfig`** (`sai_transceiver_t` *config, `sai_frame_sync_len_t` frameSyncWidth, `sai_word_width_t` bitWidth, `uint32_t` dataWordNum, `uint32_t` saiChannelMask)
Get TDM mode configurations.
- void **`SAI_GetDSPConfig`** (`sai_transceiver_t` *config, `sai_frame_sync_len_t` frameSyncWidth, `sai_word_width_t` bitWidth, `sai_mono_stereo_t` mode, `uint32_t` saiChannelMask)
Get DSP mode configurations.

Status

- static `uint32_t` **`SAI_TxGetStatusFlag`** (I2S_Type *base)
Gets the SAI Tx status flag state.
- static void **`SAI_TxClearStatusFlags`** (I2S_Type *base, `uint32_t` mask)
Clears the SAI Tx status flag state.
- static `uint32_t` **`SAI_RxGetStatusFlag`** (I2S_Type *base)
Gets the SAI Rx status flag state.
- static void **`SAI_RxClearStatusFlags`** (I2S_Type *base, `uint32_t` mask)
Clears the SAI Rx status flag state.
- void **`SAI_TxSoftwareReset`** (I2S_Type *base, `sai_reset_type_t` resetType)
Do software reset or FIFO reset .
- void **`SAI_RxSoftwareReset`** (I2S_Type *base, `sai_reset_type_t` resetType)
Do software reset or FIFO reset .
- void **`SAI_TxSetChannelFIFOMask`** (I2S_Type *base, `uint8_t` mask)
Set the Tx channel FIFO enable mask.
- void **`SAI_RxSetChannelFIFOMask`** (I2S_Type *base, `uint8_t` mask)
Set the Rx channel FIFO enable mask.
- void **`SAI_TxSetDataOrder`** (I2S_Type *base, `sai_data_order_t` order)
Set the Tx data order.
- void **`SAI_RxSetDataOrder`** (I2S_Type *base, `sai_data_order_t` order)
Set the Rx data order.
- void **`SAI_TxSetBitClockPolarity`** (I2S_Type *base, `sai_clock_polarity_t` polarity)
Set the Tx data order.
- void **`SAI_RxSetBitClockPolarity`** (I2S_Type *base, `sai_clock_polarity_t` polarity)
Set the Rx data order.

- void **SAI_TxSetFrameSyncPolarity** (I2S_Type *base, **sai_clock_polarity_t** polarity)
Set the Tx data order.
- void **SAI_RxSetFrameSyncPolarity** (I2S_Type *base, **sai_clock_polarity_t** polarity)
Set the Rx data order.
- void **SAI_TxSetFIFOPacking** (I2S_Type *base, **sai_fifo_packing_t** pack)
Set Tx FIFO packing feature.
- void **SAI_RxSetFIFOPacking** (I2S_Type *base, **sai_fifo_packing_t** pack)
Set Rx FIFO packing feature.
- static void **SAI_TxSetFIFOErrorContinue** (I2S_Type *base, bool isEnabled)
Set Tx FIFO error continue.
- static void **SAI_RxSetFIFOErrorContinue** (I2S_Type *base, bool isEnabled)
Set Rx FIFO error continue.

Interrupts

- static void **SAI_TxEnableInterrupts** (I2S_Type *base, uint32_t mask)
Enables the SAI Tx interrupt requests.
- static void **SAI_RxEnableInterrupts** (I2S_Type *base, uint32_t mask)
Enables the SAI Rx interrupt requests.
- static void **SAI_TxDisableInterrupts** (I2S_Type *base, uint32_t mask)
Disables the SAI Tx interrupt requests.
- static void **SAI_RxDisableInterrupts** (I2S_Type *base, uint32_t mask)
Disables the SAI Rx interrupt requests.

DMA Control

- static void **SAI_TxEnableDMA** (I2S_Type *base, uint32_t mask, bool enable)
Enables/disables the SAI Tx DMA requests.
- static void **SAI_RxEnableDMA** (I2S_Type *base, uint32_t mask, bool enable)
Enables/disables the SAI Rx DMA requests.
- static uintptr_t **SAI_TxGetDataRegisterAddress** (I2S_Type *base, uint32_t channel)
Gets the SAI Tx data register address.
- static uintptr_t **SAI_RxGetDataRegisterAddress** (I2S_Type *base, uint32_t channel)
Gets the SAI Rx data register address.

Bus Operations

- void **SAI_TxSetFormat** (I2S_Type *base, **sai_transfer_format_t** *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
Configures the SAI Tx audio format.
- void **SAI_RxSetFormat** (I2S_Type *base, **sai_transfer_format_t** *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
Configures the SAI Rx audio format.
- void **SAI_WriteBlocking** (I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)
Sends data using a blocking method.

- void [SAI_WriteMultiChannelBlocking](#) (I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)
Sends data to multi channel using a blocking method.
- static void [SAI_WriteData](#) (I2S_Type *base, uint32_t channel, uint32_t data)
Writes data into SAI FIFO.
- void [SAI_ReadBlocking](#) (I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)
Receives data using a blocking method.
- void [SAI_ReadMultiChannelBlocking](#) (I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)
Receives multi channel data using a blocking method.
- static uint32_t [SAI_ReadData](#) (I2S_Type *base, uint32_t channel)
Reads data from the SAI FIFO.

Transactional

- void [SAI_TransferTxCreateHandle](#) (I2S_Type *base, sai_handle_t *handle, [sai_transfer_callback_t](#) callback, void *userData)
Initializes the SAI Tx handle.
- void [SAI_TransferRxCreateHandle](#) (I2S_Type *base, sai_handle_t *handle, [sai_transfer_callback_t](#) callback, void *userData)
Initializes the SAI Rx handle.
- void [SAI_TransferTxSetConfig](#) (I2S_Type *base, sai_handle_t *handle, [sai_transceiver_t](#) *config)
SAI transmitter transfer configurations.
- void [SAI_TransferRxSetConfig](#) (I2S_Type *base, sai_handle_t *handle, [sai_transceiver_t](#) *config)
SAI receiver transfer configurations.
- status_t [SAI_TransferTxSetFormat](#) (I2S_Type *base, sai_handle_t *handle, [sai_transfer_format_t](#) *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
Configures the SAI Tx audio format.
- status_t [SAI_TransferRxSetFormat](#) (I2S_Type *base, sai_handle_t *handle, [sai_transfer_format_t](#) *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
Configures the SAI Rx audio format.
- status_t [SAI_TransferSendNonBlocking](#) (I2S_Type *base, sai_handle_t *handle, [sai_transfer_t](#) *xfer)
Performs an interrupt non-blocking send transfer on SAI.
- status_t [SAI_TransferReceiveNonBlocking](#) (I2S_Type *base, sai_handle_t *handle, [sai_transfer_t](#) *xfer)
Performs an interrupt non-blocking receive transfer on SAI.
- status_t [SAI_TransferGetSendCount](#) (I2S_Type *base, sai_handle_t *handle, size_t *count)
Gets a set byte count.
- status_t [SAI_TransferGetReceiveCount](#) (I2S_Type *base, sai_handle_t *handle, size_t *count)
Gets a received byte count.
- void [SAI_TransferAbortSend](#) (I2S_Type *base, sai_handle_t *handle)
Aborts the current send.
- void [SAI_TransferAbortReceive](#) (I2S_Type *base, sai_handle_t *handle)
Aborts the current IRQ receive.
- void [SAI_TransferTerminateSend](#) (I2S_Type *base, sai_handle_t *handle)
Terminate all SAI send.
- void [SAI_TransferTerminateReceive](#) (I2S_Type *base, sai_handle_t *handle)

- `void SAI_TransferTxHandleIRQ` (I2S_Type *base, sai_handle_t *handle)
Tx interrupt handler.
- `void SAI_TransferRxHandleIRQ` (I2S_Type *base, sai_handle_t *handle)
Rx interrupt handler.

29.4.2 Data Structure Documentation

29.4.2.1 struct sai_config_t

Data Fields

- `sai_protocol_t protocol`
Audio bus protocol in SAI.
- `sai_sync_mode_t syncMode`
SAI sync mode, control Tx/Rx clock sync.
- `bool mclkOutputEnable`
Master clock output enable, true means master clock divider enabled.
- `sai_mclk_source_t mclkSource`
Master Clock source.
- `sai_bclk_source_t bclkSource`
Bit Clock source.
- `sai_master_slave_t masterSlave`
Master or slave.

29.4.2.2 struct sai_transfer_format_t

Data Fields

- `uint32_t sampleRate_Hz`
Sample rate of audio data.
- `uint32_t bitWidth`
Data length of audio data, usually 8/16/24/32 bits.
- `sai_mono_stereo_t stereo`
Mono or stereo.
- `uint32_t masterClockHz`
Master clock frequency in Hz.
- `uint8_t watermark`
Watermark value.
- `uint8_t channel`
Transfer start channel.
- `uint8_t channelMask`
enabled channel mask value, reference _sai_channel_mask
- `uint8_t endChannel`
end channel number
- `uint8_t channelNums`
Total enabled channel numbers.
- `sai_protocol_t protocol`

- bool **isFrameSyncCompact**
Which audio protocol used.

True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.

Field Documentation

(1) **bool sai_transfer_format_t::isFrameSyncCompact**

29.4.2.3 struct sai_master_clock_t

Data Fields

- bool **mclkOutputEnable**
master clock output enable
- **sai_mclk_source_t mclkSource**
Master Clock source.
- uint32_t **mclkHz**
target mclk frequency
- uint32_t **mclkSourceClkHz**
mclk source frequency

29.4.2.4 struct sai_fifo_t

Data Fields

- bool **fifoContinueOneError**
fifo continues when error occur
- **sai_fifo_packing_t fifoPacking**
fifo packing mode
- uint8_t **fifoWatermark**
fifo watermark

29.4.2.5 struct sai_bit_clock_t

Data Fields

- bool **bclkSrcSwap**
bit clock source swap
- bool **bclkInputDelay**
bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time .
- **sai_clock_polarity_t bclkPolarity**
bit clock polarity
- **sai_bclk_source_t bclkSource**
bit Clock source

Field Documentation

(1) `bool sai_bit_clock_t::bclkInputDelay`

29.4.2.6 struct sai_frame_sync_t

Data Fields

- `uint8_t frameSyncWidth`
frame sync width in number of bit clocks
- `bool frameSyncEarly`
TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame.
- `bool frameSyncGenerateOnDemand`
internal frame sync is generated when FIFO waring flag is clear
- `sai_clock_polarity_t frameSyncPolarity`
frame sync polarity

29.4.2.7 struct sai_serial_data_t

Data Fields

- `sai_data_order_t dataOrder`
configure whether the LSB or MSB is transmitted first
- `uint8_t dataWord0Length`
configure the number of bits in the first word in each frame
- `uint8_t dataWordNLength`
configure the number of bits in the each word in each frame, except the first word
- `uint8_t dataWordLength`
used to record the data length for dma transfer
- `uint8_t dataFirstBitShifted`
Configure the bit index for the first bit transmitted for each word in the frame.
- `uint8_t dataWordNum`
configure the number of words in each frame
- `uint32_t dataMaskedWord`
configure whether the transmit word is masked

29.4.2.8 struct sai_transceiver_t

Data Fields

- `sai_serial_data_t serialData`
serial data configurations
- `sai_frame_sync_t frameSync`
ws configurations
- `sai_bit_clock_t bitClock`
bit clock configurations
- `sai_fifo_t fifo`
fifo configurations
- `sai_master_slave_t masterSlave`
transceiver is master or slave

- **sai_sync_mode_t syncMode**
transceiver sync mode
- **uint8_t startChannel**
Transfer start channel.
- **uint8_t channelMask**
enabled channel mask value, reference _sai_channel_mask
- **uint8_t endChannel**
end channel number
- **uint8_t channelNums**
Total enabled channel numbers.

29.4.2.9 struct sai_transfer_t

Data Fields

- **uint8_t * data**
Data start address to transfer.
- **size_t dataSize**
Transfer size.

Field Documentation

- (1) **uint8_t* sai_transfer_t::data**
- (2) **size_t sai_transfer_t::dataSize**

29.4.2.10 struct _sai_handle

Data Fields

- **I2S_Type * base**
base address
- **uint32_t state**
Transfer status.
- **sai_transfer_callback_t callback**
Callback function called at transfer event.
- **void * userData**
Callback parameter passed to callback function.
- **uint8_t bitWidth**
Bit width for transfer, 8/16/24/32 bits.
- **uint8_t channel**
Transfer start channel.
- **uint8_t channelMask**
enabled channel mask value, refernece _sai_channel_mask
- **uint8_t endChannel**
end channel number
- **uint8_t channelNums**
Total enabled channel numbers.
- **sai_transfer_t saiQueue [SAI_XFER_QUEUE_SIZE]**
Transfer queue storing queued transfer.

- `size_t transferSize [SAI_XFER_QUEUE_SIZE]`
Data bytes need to transfer.
- `volatile uint8_t queueUser`
Index for user to queue transfer.
- `volatile uint8_t queueDriver`
Index for driver to get the transfer data and size.
- `uint8_t watermark`
Watermark value.

29.4.3 Macro Definition Documentation

29.4.3.1 #define SAI_XFER_QUEUE_SIZE (4U)

29.4.4 Enumeration Type Documentation

29.4.4.1 anonymous enum

Enumerator

- `kStatus_SAI_TxBusy` SAI Tx is busy.
- `kStatus_SAI_RxBusy` SAI Rx is busy.
- `kStatus_SAI_TxError` SAI Tx FIFO error.
- `kStatus_SAI_RxError` SAI Rx FIFO error.
- `kStatus_SAI_QueueFull` SAI transfer queue is full.
- `kStatus_SAI_TxIdle` SAI Tx is idle.
- `kStatus_SAI_RxIdle` SAI Rx is idle.

29.4.4.2 anonymous enum

Enumerator

- `kSAI_Channel0Mask` channel 0 mask value
- `kSAI_Channel1Mask` channel 1 mask value
- `kSAI_Channel2Mask` channel 2 mask value
- `kSAI_Channel3Mask` channel 3 mask value
- `kSAI_Channel4Mask` channel 4 mask value
- `kSAI_Channel5Mask` channel 5 mask value
- `kSAI_Channel6Mask` channel 6 mask value
- `kSAI_Channel7Mask` channel 7 mask value

29.4.4.3 enum sai_protocol_t

Enumerator

- `kSAI_BusLeftJustified` Uses left justified format.

kSAI_BusRightJustified Uses right justified format.

kSAI_BusI2S Uses I2S format.

kSAI_BusPCMA Uses I2S PCM A format.

kSAI_BusPCMB Uses I2S PCM B format.

29.4.4.4 enum sai_master_slave_t

Enumerator

kSAI_Master Master mode include bclk and frame sync.

kSAI_Slave Slave mode include bclk and frame sync.

kSAI_Bclk_Master_FrameSync_Slave bclk in master mode, frame sync in slave mode

kSAI_Bclk_Slave_FrameSync_Master bclk in slave mode, frame sync in master mode

29.4.4.5 enum sai_mono_stereo_t

Enumerator

kSAI_Stereo Stereo sound.

kSAI_MonoRight Only Right channel have sound.

kSAI_MonoLeft Only left channel have sound.

29.4.4.6 enum sai_data_order_t

Enumerator

kSAI_DataLSB LSB bit transferred first.

kSAI_DataMSB MSB bit transferred first.

29.4.4.7 enum sai_clock_polarity_t

Enumerator

kSAI_PolarityActiveHigh Drive outputs on rising edge.

kSAI_PolarityActiveLow Drive outputs on falling edge.

kSAI_SampleOnFallingEdge Sample inputs on falling edge.

kSAI_SampleOnRisingEdge Sample inputs on rising edge.

29.4.4.8 enum sai_sync_mode_t

Enumerator

- kSAI_ModeAsync* Asynchronous mode.
- kSAI_ModeSync* Synchronous mode (with receiver or transmit)

29.4.4.9 enum sai_mclk_source_t

Enumerator

- kSAI_MclkSourceSysclk* Master clock from the system clock.
- kSAI_MclkSourceSelect1* Master clock from source 1.
- kSAI_MclkSourceSelect2* Master clock from source 2.
- kSAI_MclkSourceSelect3* Master clock from source 3.

29.4.4.10 enum sai_bclk_source_t

Enumerator

- kSAI_BclkSourceBusclk* Bit clock using bus clock.
- kSAI_BclkSourceMclkOption1* Bit clock MCLK option 1.
- kSAI_BclkSourceMclkOption2* Bit clock MCLK option2.
- kSAI_BclkSourceMclkOption3* Bit clock MCLK option3.
- kSAI_BclkSourceMclkDiv* Bit clock using master clock divider.
- kSAI_BclkSourceOtherSai0* Bit clock from other SAI device.
- kSAI_BclkSourceOtherSai1* Bit clock from other SAI device.

29.4.4.11 anonymous enum

Enumerator

- kSAI_WordStartInterruptEnable* Word start flag, means the first word in a frame detected.
- kSAI_SyncErrorInterruptEnable* Sync error flag, means the sync error is detected.
- kSAI_FIFOWarningInterruptEnable* FIFO warning flag, means the FIFO is empty.
- kSAI_FIFOErrorInterruptEnable* FIFO error flag.
- kSAI_FIFOResponseInterruptEnable* FIFO request, means reached watermark.

29.4.4.12 anonymous enum

Enumerator

- kSAI_FIFOWarningDMAEnable* FIFO warning caused by the DMA request.
- kSAI_FIFOResponseDMAEnable* FIFO request caused by the DMA request.

29.4.4.13 anonymous enum

Enumerator

kSAI_WordStartFlag Word start flag, means the first word in a frame detected.

kSAI_SyncErrorFlag Sync error flag, means the sync error is detected.

kSAI_FIFOErrorFlag FIFO error flag.

kSAI_FIFORequestFlag FIFO request flag.

kSAI_FIFOWarningFlag FIFO warning flag.

29.4.4.14 enum sai_reset_type_t

Enumerator

kSAI_ResetTypeSoftware Software reset, reset the logic state.

kSAI_ResetTypeFIFO FIFO reset, reset the FIFO read and write pointer.

kSAI_ResetAll All reset.

29.4.4.15 enum sai_fifo_packing_t

Enumerator

kSAI_FifoPackingDisabled Packing disabled.

kSAI_FifoPacking8bit 8 bit packing enabled

kSAI_FifoPacking16bit 16bit packing enabled

29.4.4.16 enum sai_sample_rate_t

Enumerator

kSAI_SampleRate8KHz Sample rate 8000 Hz.

kSAI_SampleRate11025Hz Sample rate 11025 Hz.

kSAI_SampleRate12KHz Sample rate 12000 Hz.

kSAI_SampleRate16KHz Sample rate 16000 Hz.

kSAI_SampleRate22050Hz Sample rate 22050 Hz.

kSAI_SampleRate24KHz Sample rate 24000 Hz.

kSAI_SampleRate32KHz Sample rate 32000 Hz.

kSAI_SampleRate44100Hz Sample rate 44100 Hz.

kSAI_SampleRate48KHz Sample rate 48000 Hz.

kSAI_SampleRate96KHz Sample rate 96000 Hz.

kSAI_SampleRate192KHz Sample rate 192000 Hz.

kSAI_SampleRate384KHz Sample rate 384000 Hz.

29.4.4.17 enum sai_word_width_t

Enumerator

- kSAI_WordWidth8bits* Audio data width 8 bits.
- kSAI_WordWidth16bits* Audio data width 16 bits.
- kSAI_WordWidth24bits* Audio data width 24 bits.
- kSAI_WordWidth32bits* Audio data width 32 bits.

29.4.4.18 enum sai_transceiver_type_t

Enumerator

- kSAI_Transmitter* sai transmitter
- kSAI_Receiver* sai receiver

29.4.4.19 enum sai_frame_sync_len_t

Enumerator

- kSAI_FrameSyncLenOneBitClk* 1 bit clock frame sync len for DSP mode
- kSAI_FrameSyncLenPerWordWidth* Frame sync length decided by word width.

29.4.5 Function Documentation

29.4.5.1 void SAI_TxInit (I2S_Type * *base*, const sai_config_t * *config*)

Deprecated Do not use this function. It has been superceded by [SAI_Init](#)

Ungates the SAI clock, resets the module, and configures SAI Tx with a configuration structure. The configuration structure can be custom filled or set with default values by [SAI_TxGetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAIM module can cause a hard fault because the clock is not enabled.

Parameters

<i>base</i>	SAI base pointer
<i>config</i>	SAI configuration structure.

29.4.5.2 void SAI_RxInit (I2S_Type * *base*, const sai_config_t * *config*)

Deprecated Do not use this function. It has been superceded by [SAI_Init](#)

Ungates the SAI clock, resets the module, and configures the SAI Rx with a configuration structure. The configuration structure can be custom filled or set with default values by [SAI_RxGetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAI module can cause a hard fault because the clock is not enabled.

Parameters

<i>base</i>	SAI base pointer
<i>config</i>	SAI configuration structure.

29.4.5.3 void SAI_TxGetDefaultConfig (sai_config_t * *config*)

Deprecated Do not use this function. It has been superceded by [SAI_GetClassicI2SConfig](#), [SAI_GetLeft-JustifiedConfig](#), [SAI_GetRightJustifiedConfig](#), [SAI_GetDSPConfig](#), [SAI_GetTDMConfig](#)

This API initializes the configuration structure for use in [SAI_TxConfig\(\)](#). The initialized structure can remain unchanged in [SAI_TxConfig\(\)](#), or it can be modified before calling [SAI_TxConfig\(\)](#). This is an example.

```
sai_config_t config;
SAI_TxGetDefaultConfig(&config);
```

Parameters

<i>config</i>	pointer to master configuration structure
---------------	---

29.4.5.4 void SAI_RxGetDefaultConfig (sai_config_t * *config*)

Deprecated Do not use this function. It has been superceded by [SAI_GetClassicI2SConfig](#), [SAI_GetLeft-JustifiedConfig](#), [SAI_GetRightJustifiedConfig](#), [SAI_GetDSPConfig](#), [SAI_GetTDMConfig](#)

This API initializes the configuration structure for use in SAI_RxConfig(). The initialized structure can remain unchanged in SAI_RxConfig() or it can be modified before calling SAI_RxConfig(). This is an example.

```
sai_config_t config;
SAI_RxGetDefaultConfig(&config);
```

Parameters

<i>config</i>	pointer to master configuration structure
---------------	---

29.4.5.5 void SAI_Init (I2S_Type * *base*)

This API gates the SAI clock. The SAI module can't operate unless SAI_Init is called to enable the clock.

Parameters

<i>base</i>	SAI base pointer.
-------------	-------------------

29.4.5.6 void SAI_Deinit (I2S_Type * *base*)

This API gates the SAI clock. The SAI module can't operate unless SAI_TxInit or SAI_RxInit is called to enable the clock.

Parameters

<i>base</i>	SAI base pointer.
-------------	-------------------

29.4.5.7 void SAI_TxReset (I2S_Type * *base*)

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

Parameters

<i>base</i>	SAI base pointer
-------------	------------------

29.4.5.8 void SAI_RxReset (I2S_Type * *base*)

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

<i>base</i>	SAI base pointer
-------------	------------------

29.4.5.9 void SAI_TxEnable (I2S_Type * *base*, bool *enable*)

Parameters

<i>base</i>	SAI base pointer.
<i>enable</i>	True means enable SAI Tx, false means disable.

29.4.5.10 void SAI_RxEnable (I2S_Type * *base*, bool *enable*)

Parameters

<i>base</i>	SAI base pointer.
<i>enable</i>	True means enable SAI Rx, false means disable.

29.4.5.11 static void SAI_TxSetBitClockDirection (I2S_Type * *base*, sai_master_slave_t *masterSlave*) [inline], [static]

Select bit clock direction, master or slave.

Parameters

<i>base</i>	SAI base pointer.
<i>masterSlave</i>	reference sai_master_slave_t.

29.4.5.12 static void SAI_RxSetBitClockDirection (I2S_Type * *base*, sai_master_slave_t *masterSlave*) [inline], [static]

Select bit clock direction, master or slave.

Parameters

<i>base</i>	SAI base pointer.
<i>masterSlave</i>	reference sai_master_slave_t.

29.4.5.13 static void SAI_RxSetFrameSyncDirection (I2S_Type * *base*, sai_master_slave_t *masterSlave*) [inline], [static]

Select frame sync direction, master or slave.

Parameters

<i>base</i>	SAI base pointer.
<i>masterSlave</i>	reference sai_master_slave_t.

29.4.5.14 static void SAI_TxSetFrameSyncDirection (I2S_Type * *base*, sai_master_slave_t *masterSlave*) [inline], [static]

Select frame sync direction, master or slave.

Parameters

<i>base</i>	SAI base pointer.
<i>masterSlave</i>	reference sai_master_slave_t.

29.4.5.15 void SAI_TxSetBitClockRate (I2S_Type * *base*, uint32_t *sourceClockHz*, uint32_t *sampleRate*, uint32_t *bitWidth*, uint32_t *channelNumbers*)

Parameters

<i>base</i>	SAI base pointer.
<i>sourceClockHz</i>	Bit clock source frequency.
<i>sampleRate</i>	Audio data sample rate.
<i>bitWidth</i>	Audio data bitWidth.
<i>channel-Numbers</i>	Audio channel numbers.

29.4.5.16 void SAI_RxSetBitClockRate (I2S_Type * *base*, uint32_t *sourceClockHz*, uint32_t *sampleRate*, uint32_t *bitWidth*, uint32_t *channelNumbers*)

Parameters

<i>base</i>	SAI base pointer.
<i>sourceClockHz</i>	Bit clock source frequency.
<i>sampleRate</i>	Audio data sample rate.
<i>bitWidth</i>	Audio data bitWidth.
<i>channel-Numbers</i>	Audio channel numbers.

29.4.5.17 void SAI_TxSetBitclockConfig (I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_bit_clock_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>masterSlave</i>	master or slave.
<i>config</i>	bit clock other configurations, can be NULL in slave mode.

29.4.5.18 void SAI_RxSetBitclockConfig (I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_bit_clock_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>masterSlave</i>	master or slave.
<i>config</i>	bit clock other configurations, can be NULL in slave mode.

29.4.5.19 void SAI_SetMasterClockConfig (I2S_Type * *base*, sai_master_clock_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	master clock configurations.

29.4.5.20 void SAI_TxSetFifoConfig (I2S_Type * *base*, sai_fifo_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	fifo configurations.

29.4.5.21 void SAI_RxSetFifoConfig (I2S_Type * *base*, sai_fifo_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	fifo configurations.

29.4.5.22 void SAI_TxSetFrameSyncConfig (I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_frame_sync_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>masterSlave</i>	master or slave.
<i>config</i>	frame sync configurations, can be NULL in slave mode.

29.4.5.23 void SAI_RxSetFrameSyncConfig (I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_frame_sync_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>masterSlave</i>	master or slave.
<i>config</i>	frame sync configurations, can be NULL in slave mode.

29.4.5.24 void SAI_TxSetSerialDataConfig (I2S_Type * *base*, sai_serial_data_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	serial data configurations.

29.4.5.25 void SAI_RxSetSerialDataConfig (I2S_Type * *base*, sai_serial_data_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	serial data configurations.

29.4.5.26 void SAI_TxSetConfig (I2S_Type * *base*, sai_transceiver_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	transmitter configurations.

29.4.5.27 void SAI_RxSetConfig (I2S_Type * *base*, sai_transceiver_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	receiver configurations.

29.4.5.28 void SAI_GetClassicI2SConfig (sai_transceiver_t * *config*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask*)

Parameters

<i>config</i>	transceiver configurations.
<i>bitWidth</i>	audio data bitWidth.
<i>mode</i>	audio data channel.
<i>saiChannelMask</i>	mask value of the channel to be enable.

29.4.5.29 **void SAI_GetLeftJustifiedConfig (*sai_transceiver_t * config, sai_word_width_t bitWidth, sai_mono_stereo_t mode, uint32_t saiChannelMask*)**

Parameters

<i>config</i>	transceiver configurations.
<i>bitWidth</i>	audio data bitWidth.
<i>mode</i>	audio data channel.
<i>saiChannel-Mask</i>	mask value of the channel to be enable.

**29.4.5.30 void SAI_GetRightJustifiedConfig (*sai_transceiver_t * config*,
sai_word_width_t bitWidth, *sai_mono_stereo_t mode*, *uint32_t saiChannelMask*)**

Parameters

<i>config</i>	transceiver configurations.
<i>bitWidth</i>	audio data bitWidth.
<i>mode</i>	audio data channel.
<i>saiChannel-Mask</i>	mask value of the channel to be enable.

29.4.5.31 void SAI_GetTDMConfig (*sai_transceiver_t * config*, *sai_frame_sync_len_t frameSyncWidth*, *sai_word_width_t bitWidth*, *uint32_t dataWordNum*, *uint32_t saiChannelMask*)

Parameters

<i>config</i>	transceiver configurations.
<i>frameSync-Width</i>	length of frame sync.
<i>bitWidth</i>	audio data word width.
<i>dataWordNum</i>	word number in one frame.
<i>saiChannel-Mask</i>	mask value of the channel to be enable.

29.4.5.32 void SAI_GetDSPConfig (sai_transceiver_t * *config*, sai_frame_sync_len_t *frameSyncWidth*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask*)

Note

DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
* SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth,
    kSAI_Stereo, channelMask)
* config->frameSync.frameSyncEarly      = true;
* SAI_TxSetConfig(base, config)
*
```

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
* SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth,
    kSAI_Stereo, channelMask)
* SAI_TxSetConfig(base, config)
*
```

Parameters

<i>config</i>	transceiver configurations.
<i>frameSyncWidth</i>	length of frame sync.
<i>bitWidth</i>	audio data bitWidth.
<i>mode</i>	audio data channel.
<i>saiChannelMask</i>	mask value of the channel to enable.

29.4.5.33 static uint32_t SAI_TxGetStatusFlag (I2S_Type * *base*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
-------------	------------------

Returns

SAI Tx status flag value. Use the Status Mask to get the status value needed.

29.4.5.34 static void SAI_TxClearStatusFlags (I2S_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	<p>State mask. It can be a combination of the following source if defined:</p> <ul style="list-style-type: none"> • kSAI_WordStartFlag • kSAI_SyncErrorFlag • kSAI_FIFOErrorFlag

29.4.5.35 static uint32_t SAI_RxGetStatusFlag (I2S_Type * *base*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
-------------	------------------

Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

29.4.5.36 static void SAI_RxClearStatusFlags (I2S_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	<p>State mask. It can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> • kSAI_WordStartFlag • kSAI_SyncErrorFlag • kSAI_FIFOErrorFlag

29.4.5.37 void SAI_TxSoftwareReset (I2S_Type * *base*, sai_reset_type_t *resetType*)

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

<i>base</i>	SAI base pointer
<i>tresetType</i>	Reset type, FIFO reset or software reset

29.4.5.38 void SAI_RxSoftwareReset (I2S_Type * *base*, sai_reset_type_t *resetType*)

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

<i>base</i>	SAI base pointer
<i>resetType</i>	Reset type, FIFO reset or software reset

29.4.5.39 void SAI_TxSetChannelFIFOMask (I2S_Type * *base*, uint8_t *mask*)

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

29.4.5.40 void SAI_RxSetChannelFIFOMask (I2S_Type * *base*, uint8_t *mask*)

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

29.4.5.41 void SAI_TxSetDataOrder (I2S_Type * *base*, sai_data_order_t *order*)

Parameters

<i>base</i>	SAI base pointer
<i>order</i>	Data order MSB or LSB

29.4.5.42 void SAI_RxSetDataOrder (I2S_Type * *base*, sai_data_order_t *order*)

Parameters

<i>base</i>	SAI base pointer
<i>order</i>	Data order MSB or LSB

29.4.5.43 void SAI_TxSetBitClockPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

<i>base</i>	SAI base pointer
<i>polarity</i>	

29.4.5.44 void SAI_RxSetBitClockPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

<i>base</i>	SAI base pointer
<i>polarity</i>	

29.4.5.45 void SAI_TxSetFrameSyncPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

<i>base</i>	SAI base pointer
<i>polarity</i>	

29.4.5.46 void SAI_RxSetFrameSyncPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

<i>base</i>	SAI base pointer
<i>polarity</i>	

29.4.5.47 void SAI_TxSetFIFOPacking (I2S_Type * *base*, sai_fifo_packing_t *pack*)

Parameters

<i>base</i>	SAI base pointer.
<i>pack</i>	FIFO pack type. It is element of sai_fifo_packing_t.

29.4.5.48 void SAI_RxSetFIFOPacking (I2S_Type * *base*, sai_fifo_packing_t *pack*)

Parameters

<i>base</i>	SAI base pointer.
<i>pack</i>	FIFO pack type. It is element of sai_fifo_packing_t.

29.4.5.49 static void SAI_TxSetFIFOErrorContinue (I2S_Type * *base*, bool *isEnabled*) [inline], [static]

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

Parameters

<i>base</i>	SAI base pointer.
<i>isEnabled</i>	Is FIFO error continue enabled, true means enable, false means disable.

29.4.5.50 static void SAI_RxSetFIFOErrorContinue (I2S_Type * *base*, bool *isEnabled*) [inline], [static]

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

<i>base</i>	SAI base pointer.
<i>isEnabled</i>	Is FIFO error continue enabled, true means enable, false means disable.

29.4.5.51 static void SAI_TxEnableInterrupts (I2S_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFOResponseInterruptEnable • kSAI_FIFOErrorInterruptEnable

29.4.5.52 static void SAI_RxEnableInterrupts (I2S_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	<p>interrupt source The parameter can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

**29.4.5.53 static void SAI_TxDisableInterrupts (I2S_Type * *base*, uint32_t *mask*)
[inline], [static]**

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	<p>interrupt source The parameter can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

**29.4.5.54 static void SAI_RxDisableInterrupts (I2S_Type * *base*, uint32_t *mask*)
[inline], [static]**

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	<p>interrupt source The parameter can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

**29.4.5.55 static void SAI_TxEnableDMA (I2S_Type * *base*, uint32_t *mask*, bool *enable*)
[inline], [static]**

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	DMA source The parameter can be combination of the following sources if defined. <ul style="list-style-type: none"> • kSAI_FIFOWarningDMAEnable • kSAI_FIFORequestDMAEnable
<i>enable</i>	True means enable DMA, false means disable DMA.

29.4.5.56 static void SAI_RxEnableDMA (I2S_Type * *base*, uint32_t *mask*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	DMA source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> • kSAI_FIFOWarningDMAEnable • kSAI_FIFORequestDMAEnable
<i>enable</i>	True means enable DMA, false means disable DMA.

29.4.5.57 static uintptr_t SAI_TxGetDataRegisterAddress (I2S_Type * *base*, uint32_t *channel*) [inline], [static]

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Which data channel used.

Returns

data register address.

29.4.5.58 static uintptr_t SAI_RxGetDataRegisterAddress (I2S_Type * *base*, uint32_t *channel*) [inline], [static]

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Which data channel used.

Returns

data register address.

29.4.5.59 void SAI_TxSetFormat (I2S_Type * *base*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

Deprecated Do not use this function. It has been superceded by [SAI_TxSetConfig](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

<i>base</i>	SAI base pointer.
<i>format</i>	Pointer to the SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If the bit clock source is a master clock, this value should equal the masterClockHz.

29.4.5.60 void SAI_RxSetFormat (I2S_Type * *base*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

Deprecated Do not use this function. It has been superceded by [SAI_RxSetConfig](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

<i>base</i>	SAI base pointer.
<i>format</i>	Pointer to the SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If the bit clock source is a master clock, this value should equal the masterClockHz.

29.4.5.61 void SAI_WriteBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be written.
<i>size</i>	Bytes to be written.

29.4.5.62 void SAI_WriteMultiChannelBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *channelMask*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>channelMask</i>	channel mask.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be written.
<i>size</i>	Bytes to be written.

29.4.5.63 static void SAI_WriteData (I2S_Type * *base*, uint32_t *channel*, uint32_t *data*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>data</i>	Data needs to be written.

29.4.5.64 void SAI_ReadBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be read.
<i>size</i>	Bytes to be read.

29.4.5.65 void SAI_ReadMultiChannelBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *channelMask*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>channelMask</i>	channel mask.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be read.
<i>size</i>	Bytes to be read.

29.4.5.66 static uint32_t SAI_ReadData (I2S_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.

Returns

Data in SAI FIFO.

29.4.5.67 void SAI_TransferTxCreateHandle (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_callback_t *callback*, void * *userData*)

This function initializes the Tx handle for the SAI Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI handle pointer.
<i>callback</i>	Pointer to the user callback function.
<i>userData</i>	User parameter passed to the callback function

29.4.5.68 void SAI_TransferRxCreateHandle (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_callback_t *callback*, void * *userData*)

This function initializes the Rx handle for the SAI Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>callback</i>	Pointer to the user callback function.
<i>userData</i>	User parameter passed to the callback function.

29.4.5.69 void SAI_TransferTxSetConfig (I2S_Type * *base*, sai_handle_t * *handle*, sai_transceiver_t * *config*)

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>config</i>	transmitter configurations.

29.4.5.70 void SAI_TransferRxSetConfig (I2S_Type * *base*, sai_handle_t * *handle*, sai_transceiver_t * *config*)

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>config</i>	receiver configurations.

29.4.5.71 status_t SAI_TransferTxSetFormat (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

Deprecated Do not use this function. It has been superceded by [SAI_TransferTxSetConfig](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>format</i>	Pointer to the SAI audio data format structure.
<i>mclkSourceClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSourceClockHz</i>	SAI bit clock source frequency in Hz. If a bit clock source is a master clock, this value should equal the masterClockHz in format.

Returns

Status of this function. Return value is the status_t.

29.4.5.72 status_t SAI_TransferRxSetFormat (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

Deprecated Do not use this function. It has been superceded by [SAI_TransferRxSetConfig](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>format</i>	Pointer to the SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If a bit clock source is a master clock, this value should equal the masterClockHz in format.

Returns

Status of this function. Return value is one of status_t.

29.4.5.73 status_t SAI_TransferSendNonBlocking (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_t * *xfer*)

Note

This API returns immediately after the transfer initiates. Call the SAI_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.
<i>xfer</i>	Pointer to the sai_transfer_t structure.

Return values

<i>kStatus_Success</i>	Successfully started the data receive.
<i>kStatus_SAI_TxBusy</i>	Previous receive still not finished.
<i>kStatus_InvalidArgument</i>	The input parameter is invalid.

29.4.5.74 status_t SAI_TransferReceiveNonBlocking (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_t * *xfer*)

Note

This API returns immediately after the transfer initiates. Call the SAI_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not *kStatus_SAI_Busy*, the transfer is finished.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	Pointer to the <i>sai_handle_t</i> structure which stores the transfer state.
<i>xfer</i>	Pointer to the <i>sai_transfer_t</i> structure.

Return values

<i>kStatus_Success</i>	Successfully started the data receive.
<i>kStatus_SAI_RxBusy</i>	Previous receive still not finished.
<i>kStatus_InvalidArgument</i>	The input parameter is invalid.

29.4.5.75 status_t SAI_TransferGetSendCount (I2S_Type * *base*, sai_handle_t * *handle*, size_t * *count*)

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the <i>sai_handle_t</i> structure which stores the transfer state.
<i>count</i>	Bytes count 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.

29.4.5.76 **status_t SAI_TransferGetReceiveCount (I2S_Type * *base*, sai_handle_t * *handle*, size_t * *count*)**

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.
<i>count</i>	Bytes count 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.

29.4.5.77 **void SAI_TransferAbortSend (I2S_Type * *base*, sai_handle_t * *handle*)**

Note

This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.

29.4.5.78 **void SAI_TransferAbortReceive (I2S_Type * *base*, sai_handle_t * *handle*)**

Note

This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.

29.4.5.79 void SAI_TransferTerminateSend (I2S_Type * *base*, sai_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSend.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

29.4.5.80 void SAI_TransferTerminateReceive (I2S_Type * *base*, sai_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceive.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

29.4.5.81 void SAI_TransferTxHandleIRQ (I2S_Type * *base*, sai_handle_t * *handle*)

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure.

29.4.5.82 void SAI_TransferRxHandleIRQ (I2S_Type * *base*, sai_handle_t * *handle*)

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure.

29.5 SAI EDMA Driver

29.5.1 Overview

Data Structures

- struct `sai_edma_handle_t`
SAI DMA transfer handle, users should not touch the content of the handle. [More...](#)

Typedefs

- typedef void(* `sai_edma_callback_t`)(I2S_Type *base, sai_edma_handle_t *handle, `status_t` status, void *userData)
SAI eDMA transfer callback function for finish and error.

Driver version

- #define `FSL_SAI_EDMA_DRIVER_VERSION` (`MAKE_VERSION(2, 5, 0)`)
Version 2.5.0.

eDMA Transactional

- void `SAI_TransferTxCreateHandleEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_edma_callback_t` callback, void *userData, `edma_handle_t` *txDmaHandle)
Initializes the SAI eDMA handle.
- void `SAI_TransferRxCreateHandleEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_edma_callback_t` callback, void *userData, `edma_handle_t` *rxDmaHandle)
Initializes the SAI Rx eDMA handle.
- void `SAI_TransferTxSetFormatEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_transfer_format_t` *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
Configures the SAI Tx audio format.
- void `SAI_TransferRxSetFormatEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_transfer_format_t` *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
Configures the SAI Rx audio format.
- void `SAI_TransferTxSetConfigEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_transceiver_t` *saiConfig)
Configures the SAI Tx.
- void `SAI_TransferRxSetConfigEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_transceiver_t` *saiConfig)
Configures the SAI Rx.
- `status_t SAI_TransferSendEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_transfer_t` *xfer)
Performs a non-blocking SAI transfer using DMA.
- `status_t SAI_TransferReceiveEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_transfer_t` *xfer)

- *Performs a non-blocking SAI receive using eDMA.*
- **status_t SAI_TransferSendLoopEDMA** (I2S_Type *base, sai_edma_handle_t *handle, sai_transfer_t *xfer, uint32_t loopTransferCount)
 - Performs a non-blocking SAI loop transfer using eDMA.*
- **status_t SAI_TransferReceiveLoopEDMA** (I2S_Type *base, sai_edma_handle_t *handle, sai_transfer_t *xfer, uint32_t loopTransferCount)
 - Performs a non-blocking SAI loop transfer using eDMA.*
- **void SAI_TransferTerminateSendEDMA** (I2S_Type *base, sai_edma_handle_t *handle)
 - Terminate all SAI send.*
- **void SAI_TransferTerminateReceiveEDMA** (I2S_Type *base, sai_edma_handle_t *handle)
 - Terminate all SAI receive.*
- **void SAI_TransferAbortSendEDMA** (I2S_Type *base, sai_edma_handle_t *handle)
 - Aborts a SAI transfer using eDMA.*
- **void SAI_TransferAbortReceiveEDMA** (I2S_Type *base, sai_edma_handle_t *handle)
 - Aborts a SAI receive using eDMA.*
- **status_t SAI_TransferGetSendCountEDMA** (I2S_Type *base, sai_edma_handle_t *handle, size_t *count)
 - Gets byte count sent by SAI.*
- **status_t SAI_TransferGetReceiveCountEDMA** (I2S_Type *base, sai_edma_handle_t *handle, size_t *count)
 - Gets byte count received by SAI.*
- **uint32_t SAI_TransferGetValidTransferSlotsEDMA** (I2S_Type *base, sai_edma_handle_t *handle)
 - Gets valid transfer slot.*

29.5.2 Data Structure Documentation

29.5.2.1 struct sai_edma_handle

Data Fields

- **edma_handle_t * dmaHandle**
 - DMA handler for SAI send.*
- **uint8_t nbytes**
 - eDMA minor byte transfer count initially configured.*
- **uint8_t bytesPerFrame**
 - Bytes in a frame.*
- **uint8_t channelMask**
 - Enabled channel mask value, reference _sai_channel_mask.*
- **uint8_t channelNums**
 - total enabled channel nums*
- **uint8_t channel**
 - Which data channel.*
- **uint8_t count**
 - The transfer data count in a DMA request.*
- **uint32_t state**
 - Internal state for SAI eDMA transfer.*
- **sai_edma_callback_t callback**
 - Callback for users while transfer finish or error occurs.*
- **void * userData**

- *User callback parameter.*
- `uint8_t tcd [(SAI_XFER_QUEUE_SIZE+1U)*sizeof(edma_tcd_t)]`
TCD pool for eDMA transfer.
- `sai_transfer_t saiQueue [SAI_XFER_QUEUE_SIZE]`
Transfer queue storing queued transfer.
- `size_t transferSize [SAI_XFER_QUEUE_SIZE]`
Data bytes need to transfer.
- `volatile uint8_t queueUser`
Index for user to queue transfer.
- `volatile uint8_t queueDriver`
Index for driver to get the transfer data and size.

Field Documentation

- (1) `uint8_t sai_edma_handle_t::nbytes`
- (2) `uint8_t sai_edma_handle_t::tcd[(SAI_XFER_QUEUE_SIZE+1U)*sizeof(edma_tcd_t)]`
- (3) `sai_transfer_t sai_edma_handle_t::saiQueue[SAI_XFER_QUEUE_SIZE]`
- (4) `volatile uint8_t sai_edma_handle_t::queueUser`

29.5.3 Function Documentation

29.5.3.1 void SAI_TransferTxCreateHandleEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_edma_callback_t *callback*, void * *userData*, edma_handle_t * *txDmaHandle*)

This function initializes the SAI master DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>base</i>	SAI peripheral base address.
<i>callback</i>	Pointer to user callback function.
<i>userData</i>	User parameter passed to the callback function.
<i>txDmaHandle</i>	eDMA handle pointer, this handle shall be static allocated by users.

29.5.3.2 void SAI_TransferRxCreateHandleEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_edma_callback_t *callback*, void * *userData*, edma_handle_t * *rxDmaHandle*)

This function initializes the SAI slave DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>base</i>	SAI peripheral base address.
<i>callback</i>	Pointer to user callback function.
<i>userData</i>	User parameter passed to the callback function.
<i>rxDmaHandle</i>	eDMA handle pointer, this handle shall be static allocated by users.

29.5.3.3 void SAI_TransferTxSetFormatEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

Deprecated Do not use this function. It has been superceded by [SAI_TransferTxSetConfigEDMA](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>format</i>	Pointer to SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If bit clock source is master clock, this value should equals to masterClockHz in format.

Return values

<i>kStatus_Success</i>	Audio format set successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

29.5.3.4 void SAI_TransferRxSetFormatEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

Deprecated Do not use this function. It has been superceded by [SAI_TransferRxSetConfigEDMA](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>format</i>	Pointer to SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If a bit clock source is the master clock, this value should equal to masterClockHz in format.

Return values

<i>kStatus_Success</i>	Audio format set successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

29.5.3.5 void SAI_TransferTxSetConfigEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transceiver_t * *saiConfig*)

Note

SAI eDMA supports data transfer in multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of *sai_transceiver_t* with the corresponding values of *_sai_channel_mask* enum, enable the FIFO Combine mode by assigning *kSAI_FifoCombineModeEnabledOnWrite* to the *fifoCombine* member of *sai_fifo_combine_t* which is a member of *sai_transceiver_t*. This is an example of multi-channel data transfer configuration step.

```
*   sai_transceiver_t config;
*   SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits,
*                           kSAI_Stereo, kSAI_Channel0Mask|kSAI_Channel1Mask);
*   config fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnWrite;
*   SAI_TransferTxSetConfigEDMA(I2S0, &edmaHandle, &config);
*
```

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>saiConfig</i>	sai configurations.

29.5.3.6 void SAI_TransferRxSetConfigEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transceiver_t * *saiConfig*)

Note

SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of *sai_transceiver_t* with the corresponding values of _sai_channel_mask enum, enable the FIFO Combine mode by assigning kSAI_FifoCombineModeEnabledOnRead to the fifoCombine member of *sai_fifo_combine_t* which is a member of *sai_transceiver_t*. This is an example of multi-channel data transfer configuration step.

```
*     sai_transceiver_t config;
*     SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits,
*                             kSAI_Stereo, kSAI_Channel0Mask|kSAI_Channel1Mask);
*     config fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnRead;
*     SAI_TransferRxSetConfigEDMA(I2S0, &edmaHandle, &config);
*
```

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>saiConfig</i>	sai configurations.

29.5.3.7 status_t SAI_TransferSendEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_t * *xfer*)

Note

This interface returns immediately after the transfer initiates. Call SAI_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

This function support multi channel transfer,

1. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers
2. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>xfer</i>	Pointer to the DMA transfer structure.

Return values

<i>kStatus_Success</i>	Start a SAI eDMA send successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.
<i>kStatus_TxBusy</i>	SAI is busy sending data.

29.5.3.8 status_t SAI_TransferReceiveEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_t * *xfer*)

Note

This interface returns immediately after the transfer initiates. Call the SAI_GetReceiveRemainingBytes to poll the transfer status and check whether the SAI transfer is finished.

This function support multi channel transfer,

1. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers
2. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI eDMA handle pointer.
<i>xfer</i>	Pointer to DMA transfer structure.

Return values

<i>kStatus_Success</i>	Start a SAI eDMA receive successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

<i>kStatus_RxBusy</i>	SAI is busy receiving data.
-----------------------	-----------------------------

29.5.3.9 status_t SAI_TransferSendLoopEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_t * *xfer*, uint32_t *loopTransferCount*)

Note

This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size.

Once the loop transfer start, application can use function SAI_TransferAbortSendEDMA to stop the loop transfer.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>xfer</i>	Pointer to the DMA transfer structure, should be a array with elements counts ≥ 1 (loopTransferCount).
<i>loopTransfer-Count</i>	the counts of xfer array.

Return values

<i>kStatus_Success</i>	Start a SAI eDMA send successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

29.5.3.10 status_t SAI_TransferReceiveLoopEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_t * *xfer*, uint32_t *loopTransferCount*)

Note

This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size.

Once the loop transfer start, application can use function SAI_TransferAbortReceiveEDMA to stop the loop transfer.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>xfer</i>	Pointer to the DMA transfer structure, should be a array with elements counts ≥ 1 (loopTransferCount).
<i>loopTransfer-Count</i>	the counts of xfer array.

Return values

<i>kStatus_Success</i>	Start a SAI eDMA receive successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

29.5.3.11 void SAI_TransferTerminateSendEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSendEDMA.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

29.5.3.12 void SAI_TransferTerminateReceiveEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceiveEDMA.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

29.5.3.13 void SAI_TransferAbortSendEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateSendEDMA.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

29.5.3.14 void SAI_TransferAbortReceiveEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateReceiveEDMA.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI eDMA handle pointer.

29.5.3.15 status_t SAI_TransferGetSendCountEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, size_t * *count*)

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>count</i>	Bytes count sent by SAI.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferIn-Progress</i>	There is no non-blocking transaction in progress.

29.5.3.16 status_t SAI_TransferGetReceiveCountEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, size_t * *count*)

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI eDMA handle pointer.
<i>count</i>	Bytes count received by SAI.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferIn-Progress</i>	There is no non-blocking transaction in progress.

29.5.3.17 `uint32_t SAI_TransferGetValidTransferSlotsEDMA (I2S_Type * base, sai_edma_handle_t * handle)`

This function can be used to query the valid transfer request slot that the application can submit. It should be called in the critical section, that means the application could call it in the corresponding callback function or disable IRQ before calling it in the application, otherwise, the returned value may not correct.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI eDMA handle pointer.

Return values

<i>valid</i>	slot count that application submit.
--------------	-------------------------------------

Chapter 30

SIM: System Integration Module Driver

30.1 Overview

The MCUXpresso SDK provides a peripheral driver for the System Integration Module (SIM) of MCUXpresso SDK devices.

Data Structures

- struct `sim_uid_t`
Unique ID. [More...](#)

Enumerations

- enum `_sim_usb_volt_reg_enable_mode` {
 `kSIM_UsbVoltRegEnable` = (int)`SIM_SOPT1_USBREGEN_MASK`,
 `kSIM_UsbVoltRegEnableInLowPower` = `SIM_SOPT1_USBVSTBY_MASK`,
 `kSIM_UsbVoltRegEnableInStop` = `SIM_SOPT1_USBSSTBY_MASK`,
 `kSIM_UsbVoltRegEnableInAllModes` }
USB voltage regulator enable setting.
- enum `_sim_flash_mode` {
 `kSIM_FlashDisableInWait` = `SIM_FCFG1_FLASHDOZE_MASK`,
 `kSIM_FlashDisable` = `SIM_FCFG1_FLASHDIS_MASK` }
Flash enable mode.

Functions

- void `SIM_SetUsbVoltRegulatorEnableMode` (uint32_t mask)
Sets the USB voltage regulator setting.
- void `SIM_GetUniqueId` (`sim_uid_t` *uid)
Gets the unique identification register value.
- static void `SIM_SetFlashMode` (uint8_t mode)
Sets the flash enable mode.

Driver version

- #define `FSL_SIM_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 3)`)

30.2 Data Structure Documentation

30.2.1 struct `sim_uid_t`

Data Fields

- `uint32_t H`

- `uint32_t MH`
UIDH.
- `uint32_t ML`
UIDML.
- `uint32_t L`
UIDL.

Field Documentation

- (1) `uint32_t sim_uid_t::H`
- (2) `uint32_t sim_uid_t::MH`
- (3) `uint32_t sim_uid_t::ML`
- (4) `uint32_t sim_uid_t::L`

30.3 Enumeration Type Documentation

30.3.1 enum _sim_usb_volt_reg_enable_mode

Enumerator

`kSIM_UsbVoltRegEnable` Enable voltage regulator.

`kSIM_UsbVoltRegEnableInLowPower` Enable voltage regulator in VLPR/VLPW modes.

`kSIM_UsbVoltRegEnableInStop` Enable voltage regulator in STOP/VLPS/LLS/VLLS modes.

`kSIM_UsbVoltRegEnableInAllModes` Enable voltage regulator in all power modes.

30.3.2 enum _sim_flash_mode

Enumerator

`kSIM_FlashDisableInWait` Disable flash in wait mode.

`kSIM_FlashDisable` Disable flash in normal mode.

30.4 Function Documentation

30.4.1 void SIM_SetUsbVoltRegulatorEnableMode (`uint32_t mask`)

This function configures whether the USB voltage regulator is enabled in normal RUN mode, STOP-VLPS/LLS/VLLS modes, and VLPR/VLPW modes. The configurations are passed in as mask value of `_sim_usb_volt_reg_enable_mode`. For example, to enable USB voltage regulator in RUN/VLPR/VLPW modes and disable in STOP/VLPS/LLS/VLLS mode, use:

```
SIM_SetUsbVoltRegulatorEnableMode(kSIM_UsbVoltRegEnable | kSIM_UsbVoltRegEnableInLowPower);
```

Parameters

<i>mask</i>	USB voltage regulator enable setting.
-------------	---------------------------------------

30.4.2 void SIM_GetUniqueId (sim_uid_t * *uid*)

Parameters

<i>uid</i>	Pointer to the structure to save the UID value.
------------	---

30.4.3 static void SIM_SetFlashMode (uint8_t *mode*) [inline], [static]

Parameters

<i>mode</i>	The mode to set; see _sim_flash_mode for mode details.
-------------	--

Chapter 31

SMC: System Mode Controller Driver

31.1 Overview

The MCUXpresso SDK provides a peripheral driver for the System Mode Controller (SMC) module of MCUXpresso SDK devices. The SMC module sequences the system in and out of all low-power stop and run modes.

API functions are provided to configure the system for working in a dedicated power mode. For different power modes, `SMC_SetPowerModeXXX()` function accepts different parameters. System power mode state transitions are not available between power modes. For details about available transitions, see the power mode transitions section in the SoC reference manual.

31.2 Typical use case

31.2.1 Enter wait or stop modes

SMC driver provides APIs to set MCU to different wait modes and stop modes. Pre and post functions are used for setting the modes. The pre functions and post functions are used as follows.

Disable/enable the interrupt through PRIMASK. This is an example use case. The application sets the wakeup interrupt and calls SMC function `SMC_SetPowerModeStop` to set the MCU to STOP mode, but the wakeup interrupt happens so quickly that the ISR completes before the function `SMC_SetPowerModeStop`. As a result, the MCU enters the STOP mode and never is woken up by the interrupt. In this use case, the application first disables the interrupt through PRIMASK, sets the wakeup interrupt, and enters the STOP mode. After wakeup, enable the interrupt through PRIMASK. The MCU can still be woken up by disabling the interrupt through PRIMASK. The pre and post functions handle the PRIMASK.

```
SMC_PreEnterStopModes();  
/* Enable the wakeup interrupt here. */  
SMC_SetPowerModeStop(SMC, kSMC_PartialStop);  
SMC_PostExitStopModes();
```

For legacy Kinetis, when entering stop modes, the flash speculation might be interrupted. As a result, the prefetched code or data might be broken. To make sure the flash is idle when entering the stop modes, smc driver allocates a RAM region, the code to enter stop modes are executed in RAM, thus the flash is idle and no prefetch is performed while entering stop modes. Application should make sure that, the rw data of `fsl_smci.c` is located in memory region which is not powered off in stop modes, especially LLS2 modes.

For STOP, VLPS, and LLS3, the whole RAM are powered up, so after woken up, the RAM function could continue executing. For VLLS mode, the system resets after woken up, the RAM content might be re-initialized. For LLS2 mode, only part of RAM are powered on, so application must make sure that, the

rw data of fsl_smcc.c is located in memory region which is not powered off, otherwise after woken up, the MCU could not get right code to execute.

Data Structures

- struct `smc_power_mode_lls_config_t`
`SMC Low-Leakage Stop power mode configuration.` [More...](#)
- struct `smc_power_mode_vlls_config_t`
`SMC Very Low-Leakage Stop power mode configuration.` [More...](#)

Enumerations

- enum `smc_power_mode_protection_t` {

`kSMC_AllowPowerModeVlls` = SMC_PMPROT_AVLLS_MASK,
`kSMC_AllowPowerModeLls` = SMC_PMPROT_ALLS_MASK,
`kSMC_AllowPowerModeVlp` = SMC_PMPROT_AVLP_MASK,
`kSMC_AllowPowerModeHsrun` = SMC_PMPROT_AHSRUN_MASK,
`kSMC_AllowPowerModeAll` }

Power Modes Protection.
- enum `smc_power_state_t` {

`kSMC_PowerStateRun` = 0x01U << 0U,
`kSMC_PowerStateStop` = 0x01U << 1U,
`kSMC_PowerStateVlpr` = 0x01U << 2U,
`kSMC_PowerStateVlpw` = 0x01U << 3U,
`kSMC_PowerStateVlps` = 0x01U << 4U,
`kSMC_PowerStateLls` = 0x01U << 5U,
`kSMC_PowerStateVlls` = 0x01U << 6U,
`kSMC_PowerStateHsrun` = 0x01U << 7U }

Power Modes in PMSTAT.
- enum `smc_run_mode_t` {

`kSMC_RunNormal` = 0U,
`kSMC_RunVlpr` = 2U,
`kSMC_Hsrun` = 3U }

Run mode definition.
- enum `smc_stop_mode_t` {

`kSMC_StopNormal` = 0U,
`kSMC_StopVlps` = 2U,
`kSMC_StopLls` = 3U,
`kSMC_StopVlls` = 4U }

Stop mode definition.
- enum `smc_stop_submode_t` {

`kSMC_StopSub0` = 0U,
`kSMC_StopSub1` = 1U,
`kSMC_StopSub2` = 2U,
`kSMC_StopSub3` = 3U }

VLLS/LLS stop sub mode definition.

- enum `smc_partial_stop_option_t` {

 `kSMC_PartialStop` = 0U,

 `kSMC_PartialStop1` = 1U,

 `kSMC_PartialStop2` = 2U }

 Partial STOP option.
- enum { `kStatus_SMC_StopAbort` = MAKE_STATUS(kStatusGroup_POWER, 0) }

 _smc_status, SMC configuration status.

Driver version

- #define `FSL_SMC_DRIVER_VERSION` (MAKE_VERSION(2, 0, 7))

 SMC driver version.

System mode controller APIs

- static void `SMC_SetPowerModeProtection` (SMC_Type *base, uint8_t allowedModes)

 Configures all power mode protection settings.
- static `smc_power_state_t SMC_GetPowerModeState` (SMC_Type *base)

 Gets the current power mode status.
- void `SMC_PreEnterStopModes` (void)

 Prepares to enter stop modes.
- void `SMC_PostExitStopModes` (void)

 Recoveries after wake up from stop modes.
- void `SMC_PreEnterWaitModes` (void)

 Prepares to enter wait modes.
- void `SMC_PostExitWaitModes` (void)

 Recoveries after wake up from wait modes.
- `status_t SMC_SetPowerModeRun` (SMC_Type *base)

 Configures the system to RUN power mode.
- `status_t SMC_SetPowerModeHsrun` (SMC_Type *base)

 Configures the system to HSRUN power mode.
- `status_t SMC_SetPowerModeWait` (SMC_Type *base)

 Configures the system to WAIT power mode.
- `status_t SMC_SetPowerModeStop` (SMC_Type *base, `smc_partial_stop_option_t` option)

 Configures the system to Stop power mode.
- `status_t SMC_SetPowerModeVlpr` (SMC_Type *base)

 Configures the system to VLPR power mode.
- `status_t SMC_SetPowerModeVlpw` (SMC_Type *base)

 Configures the system to VLPW power mode.
- `status_t SMC_SetPowerModeVlps` (SMC_Type *base)

 Configures the system to VLPS power mode.
- `status_t SMC_SetPowerModeLls` (SMC_Type *base, const `smc_power_mode_lls_config_t` *config)

 Configures the system to LLS power mode.
- `status_t SMC_SetPowerModeVlls` (SMC_Type *base, const `smc_power_mode_vlls_config_t` *config)

 Configures the system to VLLS power mode.

31.3 Data Structure Documentation

31.3.1 struct smc_power_mode_lls_config_t

Data Fields

- [smc_stop_submode_t subMode](#)
Low-leakage Stop sub-mode.

31.3.2 struct smc_power_mode_vlls_config_t

Data Fields

- [smc_stop_submode_t subMode](#)
Very Low-leakage Stop sub-mode.
- [bool enablePorDetectInVlls0](#)
Enable Power on reset detect in VLLS mode.

31.4 Enumeration Type Documentation

31.4.1 enum smc_power_mode_protection_t

Enumerator

- kSMC_AllowPowerModeVlls*** Allow Very-low-leakage Stop Mode.
- kSMC_AllowPowerModeLls*** Allow Low-leakage Stop Mode.
- kSMC_AllowPowerModeVlp*** Allow Very-Low-power Mode.
- kSMC_AllowPowerModeHsrun*** Allow High-speed Run mode.
- kSMC_AllowPowerModeAll*** Allow all power mode.

31.4.2 enum smc_power_state_t

Enumerator

- kSMC_PowerStateRun*** 0000_0001 - Current power mode is RUN
- kSMC_PowerStateStop*** 0000_0010 - Current power mode is STOP
- kSMC_PowerStateVlpr*** 0000_0100 - Current power mode is VLPR
- kSMC_PowerStateVlpw*** 0000_1000 - Current power mode is VLPW
- kSMC_PowerStateVlps*** 0001_0000 - Current power mode is VLPS
- kSMC_PowerStateLls*** 0010_0000 - Current power mode is LLS
- kSMC_PowerStateVlls*** 0100_0000 - Current power mode is VLLS
- kSMC_PowerStateHsrun*** 1000_0000 - Current power mode is HSRUN

31.4.3 enum smc_run_mode_t

Enumerator

- kSMC_RunNormal* Normal RUN mode.
- kSMC_RunVlpr* Very-low-power RUN mode.
- kSMC_Hsrun* High-speed Run mode (HSRUN).

31.4.4 enum smc_stop_mode_t

Enumerator

- kSMC_StopNormal* Normal STOP mode.
- kSMC_StopVlps* Very-low-power STOP mode.
- kSMC_StopLls* Low-leakage Stop mode.
- kSMC_StopVlls* Very-low-leakage Stop mode.

31.4.5 enum smc_stop_submode_t

Enumerator

- kSMC_StopSub0* Stop submode 0, for VLLS0/LLS0.
- kSMC_StopSub1* Stop submode 1, for VLLS1/LLS1.
- kSMC_StopSub2* Stop submode 2, for VLLS2/LLS2.
- kSMC_StopSub3* Stop submode 3, for VLLS3/LLS3.

31.4.6 enum smc_partial_stop_option_t

Enumerator

- kSMC_PartialStop* STOP - Normal Stop mode.
- kSMC_PartialStop1* Partial Stop with both system and bus clocks disabled.
- kSMC_PartialStop2* Partial Stop with system clock disabled and bus clock enabled.

31.4.7 anonymous enum

Enumerator

- kStatus_SMC_StopAbort* Entering Stop mode is abort.

31.5 Function Documentation

31.5.1 static void SMC_SetPowerModeProtection (**SMC_Type** * *base*, **uint8_t** *allowedModes*) [inline], [static]

This function configures the power mode protection settings for supported power modes in the specified chip family. The available power modes are defined in the smc_power_mode_protection_t. This should be done at an early system level initialization stage. See the reference manual for details. This register can only write once after the power reset.

The allowed modes are passed as bit map. For example, to allow LLS and VLLS, use SMC_SetPowerModeProtection(kSMC_AllowPowerModeVlls | kSMC_AllowPowerModeVlps). To allow all modes, use SMC_SetPowerModeProtection(kSMC_AllowPowerModeAll).

Parameters

<i>base</i>	SMC peripheral base address.
<i>allowedModes</i>	Bitmap of the allowed power modes.

31.5.2 static smc_power_state_t SMC_GetPowerModeState (**SMC_Type** * *base*) [inline], [static]

This function returns the current power mode status. After the application switches the power mode, it should always check the status to check whether it runs into the specified mode or not. The application should check this mode before switching to a different mode. The system requires that only certain modes can switch to other specific modes. See the reference manual for details and the smc_power_state_t for information about the power status.

Parameters

<i>base</i>	SMC peripheral base address.
-------------	------------------------------

Returns

Current power mode status.

31.5.3 void SMC_PreEnterStopModes (void)

This function should be called before entering STOP/VLPS/LLS/VLLS modes.

31.5.4 void SMC_PostExitStopModes (void)

This function should be called after wake up from STOP/VLPS/LLS/VLLS modes. It is used with [SMC_PreEnterStopModes](#).

31.5.5 void SMC_PreEnterWaitModes (void)

This function should be called before entering WAIT/VLPW modes.

31.5.6 void SMC_PostExitWaitModes (void)

This function should be called after wake up from WAIT/VLPW modes. It is used with [SMC_PreEnterWaitModes](#).

31.5.7 status_t SMC_SetPowerModeRun (SMC_Type * *base*)

Parameters

<i>base</i>	SMC peripheral base address.
-------------	------------------------------

Returns

SMC configuration error code.

31.5.8 status_t SMC_SetPowerModeHsrun (SMC_Type * *base*)

Parameters

<i>base</i>	SMC peripheral base address.
-------------	------------------------------

Returns

SMC configuration error code.

31.5.9 status_t SMC_SetPowerModeWait (SMC_Type * *base*)

Parameters

<i>base</i>	SMC peripheral base address.
-------------	------------------------------

Returns

SMC configuration error code.

31.5.10 status_t SMC_SetPowerModeStop (SMC_Type * *base*, smc_partial_stop_option_t *option*)

Parameters

<i>base</i>	SMC peripheral base address.
<i>option</i>	Partial Stop mode option.

Returns

SMC configuration error code.

31.5.11 status_t SMC_SetPowerModeVlpr (SMC_Type * *base*)

Parameters

<i>base</i>	SMC peripheral base address.
-------------	------------------------------

Returns

SMC configuration error code.

31.5.12 status_t SMC_SetPowerModeVlpw (SMC_Type * *base*)

Parameters

<i>base</i>	SMC peripheral base address.
-------------	------------------------------

Returns

SMC configuration error code.

31.5.13 status_t SMC_SetPowerModeVlps (SMC_Type * *base*)

Parameters

<i>base</i>	SMC peripheral base address.
-------------	------------------------------

Returns

SMC configuration error code.

31.5.14 status_t SMC_SetPowerModeLls (SMC_Type * *base*, const smc_power_mode_lls_config_t * *config*)

Parameters

<i>base</i>	SMC peripheral base address.
<i>config</i>	The LLS power mode configuration structure

Returns

SMC configuration error code.

31.5.15 status_t SMC_SetPowerModeVlls (SMC_Type * *base*, const smc_power_mode_vlls_config_t * *config*)

Parameters

<i>base</i>	SMC peripheral base address.
<i>config</i>	The VLLS power mode configuration structure.

Returns

SMC configuration error code.

Chapter 32

UART: Universal Asynchronous Receiver/Transmitter Driver

32.1 Overview

Modules

- [UART CMSIS Driver](#)
- [UART Driver](#)
- [UART FreeRTOS Driver](#)
- [UART eDMA Driver](#)

32.2 UART Driver

32.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Universal Asynchronous Receiver/Transmitter (UART) module of MCUXpresso SDK devices.

The UART driver includes functional APIs and transactional APIs.

Functional APIs are used for UART initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the UART peripheral and how to organize functional APIs to meet the application requirements. All functional APIs use the peripheral base address as the first parameter. UART functional operation groups provide the functional API 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 `uart_handle_t` as the second parameter. Initialize the handle by calling the [UART_TransferCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer, which means that the functions [UART_TransferSendNonBlocking\(\)](#) and [UART_TransferReceiveNonBlocking\(\)](#) set up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_UART_TxIdle` and `kStatus_UART_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 [UART_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 [UART_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_UART_RxIdle`.

If the receive ring buffer is full, the upper layer is informed through a callback with the `kStatus_UART_RxRingBufferOverrun`. In the callback function, the upper layer reads data out from the ring buffer. If not, existing 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.

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/uart`. In this example, the buffer size is 32, but only 31 bytes are used for saving data.

32.2.2 Typical use case

32.2.2.1 UART Send/receive using a polling method

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

32.2.2.2 UART Send/receive using an interrupt method

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

32.2.2.3 UART Receive using the ringbuffer feature

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

32.2.2.4 UART Send/Receive using the DMA method

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

Data Structures

- struct [uart_config_t](#)
UART configuration structure. [More...](#)
- struct [uart_transfer_t](#)
UART transfer structure. [More...](#)
- struct [uart_handle_t](#)
UART handle structure. [More...](#)

Macros

- #define [UART_RETRY_TIMES](#) 0U /* Defining to zero means to keep waiting for the flag until it is assert/deassert. */
Retry times for waiting flag.

TypeDefs

- typedef void(* [uart_transfer_callback_t](#))(UART_Type *base, uart_handle_t *handle, [status_t](#) status, void *userData)
UART transfer callback function.

Enumerations

- enum {

kStatus_UART_TxBusy = MAKE_STATUS(kStatusGroup_UART, 0),

kStatus_UART_RxBusy = MAKE_STATUS(kStatusGroup_UART, 1),

kStatus_UART_TxIdle = MAKE_STATUS(kStatusGroup_UART, 2),

kStatus_UART_RxIdle = MAKE_STATUS(kStatusGroup_UART, 3),

kStatus_UART_TxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_UART, 4),

kStatus_UART_RxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_UART, 5),

kStatus_UART_FlagCannotClearManually,

kStatus_UART_Error = MAKE_STATUS(kStatusGroup_UART, 7),

kStatus_UART_RxRingBufferOverrun = MAKE_STATUS(kStatusGroup_UART, 8),

kStatus_UART_RxHardwareOverrun = MAKE_STATUS(kStatusGroup_UART, 9),

kStatus_UART_NoiseError = MAKE_STATUS(kStatusGroup_UART, 10),

kStatus_UART_FramingError = MAKE_STATUS(kStatusGroup_UART, 11),

kStatus_UART_ParityError = MAKE_STATUS(kStatusGroup_UART, 12),

kStatus_UART_BaudrateNotSupport,

kStatus_UART_IdleLineDetected = MAKE_STATUS(kStatusGroup_UART, 14),

kStatus_UART_Timeout = MAKE_STATUS(kStatusGroup_UART, 15) }

Error codes for the UART driver.

- enum `uart_parity_mode_t` {

kUART_ParityDisabled = 0x0U,

kUART_ParityEven = 0x2U,

kUART_ParityOdd = 0x3U }
- UART parity mode.*
- enum `uart_stop_bit_count_t` {

kUART_OneStopBit = 0U,

kUART_TwoStopBit = 1U }
- UART stop bit count.*
- enum `uart_idle_type_select_t` {

kUART_IdleTypeStartBit = 0U,

kUART_IdleTypeStopBit = 1U }
- UART idle type select.*
- enum `_uart_interrupt_enable` {

kUART_LinBreakInterruptEnable = (UART_BDH_LBKDIE_MASK),

kUART_RxActiveEdgeInterruptEnable = (UART_BDH_RXEDGIE_MASK),

kUART_TxDataRegEmptyInterruptEnable = (UART_C2_TIE_MASK << 8),

kUART_TransmissionCompleteInterruptEnable = (UART_C2_TCIE_MASK << 8),

kUART_RxDataRegFullInterruptEnable = (UART_C2_RIE_MASK << 8),

kUART_IdleLineInterruptEnable = (UART_C2_ILIE_MASK << 8),

kUART_RxOverrunInterruptEnable = (UART_C3_ORIE_MASK << 16),

kUART_NoiseErrorInterruptEnable = (UART_C3_NEIE_MASK << 16),

kUART_FramingErrorInterruptEnable = (UART_C3_FEIE_MASK << 16),

kUART_ParityErrorInterruptEnable = (UART_C3_PEIE_MASK << 16),

kUART_RxFifoOverflowInterruptEnable = (UART_CFIFO_RXOFE_MASK << 24),

kUART_TxFifoOverflowInterruptEnable = (UART_CFIFO_TXOFE_MASK << 24),

```

kUART_RxFifoUnderflowInterruptEnable = (UART_CFIFO_RXUFE_MASK << 24) }

UART interrupt configuration structure, default settings all disabled.

• enum {
    kUART_TxDataRegEmptyFlag = (UART_S1_TDRE_MASK),
    kUART_TransmissionCompleteFlag = (UART_S1_TC_MASK),
    kUART_RxDataRegFullFlag = (UART_S1_RDRF_MASK),
    kUART_IdleLineFlag = (UART_S1_IDLE_MASK),
    kUART_RxOverrunFlag = (UART_S1_OR_MASK),
    kUART_NoiseErrorFlag = (UART_S1_NF_MASK),
    kUART_FramingErrorFlag = (UART_S1_FE_MASK),
    kUART_ParityErrorFlag = (UART_S1_PF_MASK),
    kUART_LinBreakFlag,
    kUART_RxActiveEdgeFlag,
    kUART_RxActiveFlag,
    kUART_NoiseErrorInRxDataRegFlag = (UART_ED_NOISY_MASK << 16),
    kUART_ParityErrorInRxDataRegFlag = (UART_ED_PARITYE_MASK << 16),
    kUART_TxFifoEmptyFlag = (int)(UART_SFIFO_TXEMPT_MASK << 24),
    kUART_RxFifoEmptyFlag = (UART_SFIFO_RXEMPT_MASK << 24),
    kUART_TxFifoOverflowFlag = (UART_SFIFO_TXOF_MASK << 24),
    kUART_RxFifoOverflowFlag = (UART_SFIFO_RXOF_MASK << 24),
    kUART_RxFifoUnderflowFlag = (UART_SFIFO_RXUF_MASK << 24) }

UART status flags.

```

Functions

- `uint32_t UART_GetInstance (UART_Type *base)`
Get the UART instance from peripheral base address.

Variables

- `void * s_uartHandle []`
Pointers to uart handles for each instance.
- `uart_isr_t s_uartIsr`
Pointer to uart IRQ handler for each instance.

Driver version

- `#define FSL_UART_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))`
UART driver version.

Initialization and deinitialization

- `status_t UART_Init (UART_Type *base, const uart_config_t *config, uint32_t srcClock_Hz)`

- **void `UART_Deinit`** (UART_Type *base)
Deinitializes a UART instance.
- **void `UART_GetDefaultConfig`** (uart_config_t *config)
Gets the default configuration structure.
- **status_t `UART_SetBaudRate`** (UART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
Sets the UART instance baud rate.
- **void `UART_Enable9bitMode`** (UART_Type *base, bool enable)
Enable 9-bit data mode for UART.
- **static void `UART_SetMatchAddress`** (UART_Type *base, uint8_t address1, uint8_t address2)
Set the UART slave address.
- **static void `UART_EnableMatchAddress`** (UART_Type *base, bool match1, bool match2)
Enable the UART match address feature.
- **static void `UART_Set9thTransmitBit`** (UART_Type *base)
Set UART 9th transmit bit.
- **static void `UART_Clear9thTransmitBit`** (UART_Type *base)
Clear UART 9th transmit bit.

Status

- **uint32_t `UART_GetStatusFlags`** (UART_Type *base)
Gets UART status flags.
- **status_t `UART_ClearStatusFlags`** (UART_Type *base, uint32_t mask)
Clears status flags with the provided mask.

Interrupts

- **void `UART_EnableInterrupts`** (UART_Type *base, uint32_t mask)
Enables UART interrupts according to the provided mask.
- **void `UART_DisableInterrupts`** (UART_Type *base, uint32_t mask)
Disables the UART interrupts according to the provided mask.
- **uint32_t `UART_GetEnabledInterrupts`** (UART_Type *base)
Gets the enabled UART interrupts.

DMA Control

- **static uint32_t `UART_GetDataRegisterAddress`** (UART_Type *base)
Gets the UART data register address.
- **static void `UART_EnableTxDMA`** (UART_Type *base, bool enable)
Enables or disables the UART transmitter DMA request.
- **static void `UART_EnableRxDMA`** (UART_Type *base, bool enable)
Enables or disables the UART receiver DMA.

Bus Operations

- **static void `UART_EnableTx`** (UART_Type *base, bool enable)

Enables or disables the UART transmitter.

- static void **UART_EnableRx** (UART_Type *base, bool enable)

Enables or disables the UART receiver.

- static void **UART_WriteByte** (UART_Type *base, uint8_t data)

Writes to the TX register.

- static uint8_t **UART_ReadByte** (UART_Type *base)

Reads the RX register directly.

- static uint8_t **UART_GetRxFifoCount** (UART_Type *base)

Gets the rx FIFO data count.

- static uint8_t **UART_GetTxFifoCount** (UART_Type *base)

Gets the tx FIFO data count.

- void **UART_SendAddress** (UART_Type *base, uint8_t address)

Transmit an address frame in 9-bit data mode.

- status_t **UART_WriteBlocking** (UART_Type *base, const uint8_t *data, size_t length)

Writes to the TX register using a blocking method.

- status_t **UART_ReadBlocking** (UART_Type *base, uint8_t *data, size_t length)

Read RX data register using a blocking method.

Transactional

- void **UART_TransferCreateHandle** (UART_Type *base, uart_handle_t *handle, **uart_transfer_callback_t** callback, void *userData)

Initializes the UART handle.

- void **UART_TransferStartRingBuffer** (UART_Type *base, uart_handle_t *handle, uint8_t *ringBuffer, size_t ringBufferSize)

Sets up the RX ring buffer.

- void **UART_TransferStopRingBuffer** (UART_Type *base, uart_handle_t *handle)

Aborts the background transfer and uninstalls the ring buffer.

- size_t **UART_TransferGetRxRingBufferLength** (uart_handle_t *handle)

Get the length of received data in RX ring buffer.

- status_t **UART_TransferSendNonBlocking** (UART_Type *base, uart_handle_t *handle, **uart_transfer_t** *xfer)

Transmits a buffer of data using the interrupt method.

- void **UART_TransferAbortSend** (UART_Type *base, uart_handle_t *handle)

Aborts the interrupt-driven data transmit.

- status_t **UART_TransferGetSendCount** (UART_Type *base, uart_handle_t *handle, uint32_t *count)

Gets the number of bytes sent out to bus.

- status_t **UART_TransferReceiveNonBlocking** (UART_Type *base, uart_handle_t *handle, **uart_transfer_t** *xfer, size_t *receivedBytes)

Receives a buffer of data using an interrupt method.

- void **UART_TransferAbortReceive** (UART_Type *base, uart_handle_t *handle)

Aborts the interrupt-driven data receiving.

- status_t **UART_TransferGetReceiveCount** (UART_Type *base, uart_handle_t *handle, uint32_t *count)

Gets the number of bytes that have been received.

- status_t **UART_EnableTxFIFO** (UART_Type *base, bool enable)

Enables or disables the UART Tx FIFO.

- status_t **UART_EnableRxFIFO** (UART_Type *base, bool enable)

Enables or disables the UART Rx FIFO.

- static void [UART_SetRxFifoWatermark](#) (UART_Type *base, uint8_t water)
Sets the rx FIFO watermark.
- static void [UART_SetTxFifoWatermark](#) (UART_Type *base, uint8_t water)
Sets the tx FIFO watermark.
- void [UART_TransferHandleIRQ](#) (UART_Type *base, void *irqHandle)
UART IRQ handle function.
- void [UART_TransferHandleErrorIRQ](#) (UART_Type *base, void *irqHandle)
UART Error IRQ handle function.

32.2.3 Data Structure Documentation

32.2.3.1 struct uart_config_t

Data Fields

- uint32_t [baudRate_Bps](#)
UART baud rate.
- [uart_parity_mode_t](#) [parityMode](#)
Parity mode, disabled (default), even, odd.
- uint8_t [txFifoWatermark](#)
TX FIFO watermark.
- uint8_t [rxFifoWatermark](#)
RX FIFO watermark.
- bool [enableRxRTS](#)
RX RTS enable.
- bool [enableTxCTS](#)
TX CTS enable.
- [uart_idle_type_select_t](#) [idleType](#)
IDLE type select.
- bool [enableTx](#)
Enable TX.
- bool [enableRx](#)
Enable RX.

Field Documentation

(1) [uart_idle_type_select_t](#) [uart_config_t::idleType](#)

32.2.3.2 struct uart_transfer_t

Data Fields

- size_t [dataSize](#)
The byte count to be transfer.
- uint8_t * [data](#)
The buffer of data to be transfer.
- uint8_t * [rxData](#)
The buffer to receive data.

- `const uint8_t * txData`
The buffer of data to be sent.

Field Documentation

- (1) `uint8_t* uart_transfer_t::data`
- (2) `uint8_t* uart_transfer_t::rxData`
- (3) `const uint8_t* uart_transfer_t::txData`
- (4) `size_t uart_transfer_t::dataSize`

32.2.3.3 struct _uart_handle

Data Fields

- `const 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.
- `uart_transfer_callback_t callback`
Callback function.
- `void * userData`
UART callback function parameter.
- `volatile uint8_t txState`
TX transfer state.
- `volatile uint8_t rxState`
RX transfer state.

Field Documentation

- (1) `const uint8_t* volatile uart_handle_t::txData`

```
(2) volatile size_t uart_handle_t::txDataSize
(3) size_t uart_handle_t::txDataSizeAll
(4) uint8_t* volatile uart_handle_t::rxData
(5) volatile size_t uart_handle_t::rxDataSize
(6) size_t uart_handle_t::rxDataSizeAll
(7) uint8_t* uart_handle_t::rxRingBuffer
(8) size_t uart_handle_t::rxRingBufferSize
(9) volatile uint16_t uart_handle_t::rxRingBufferHead
(10) volatile uint16_t uart_handle_t::rxRingBufferTail
(11) uart_transfer_callback_t uart_handle_t::callback
(12) void* uart_handle_t::userData
(13) volatile uint8_t uart_handle_t::txState
```

32.2.4 Macro Definition Documentation

32.2.4.1 #define FSL_UART_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))

32.2.4.2 #define UART_RETRY_TIMES 0U /* Defining to zero means to keep waiting for the flag until it is assert/deassert. */

32.2.5 Typedef Documentation

32.2.5.1 **typedef void(* uart_transfer_callback_t)(UART_Type *base, uart_handle_t *handle, status_t status, void *userData)**

32.2.6 Enumeration Type Documentation

32.2.6.1 anonymous enum

Enumerator

- kStatus_UART_TxBusy* Transmitter is busy.
- kStatus_UART_RxBusy* Receiver is busy.
- kStatus_UART_TxIdle* UART transmitter is idle.
- kStatus_UART_RxIdle* UART receiver is idle.
- kStatus_UART_TxWatermarkTooLarge* TX FIFO watermark too large.

kStatus_UART_RxWatermarkTooLarge RX FIFO watermark too large.
kStatus_UART_FlagCannotClearManually UART flag can't be manually cleared.
kStatus_UART_Error Error happens on UART.
kStatus_UART_RxRingBufferOverrun UART RX software ring buffer overrun.
kStatus_UART_RxHardwareOverrun UART RX receiver overrun.
kStatus_UART_NoiseError UART noise error.
kStatus_UART_FramingError UART framing error.
kStatus_UART_ParityError UART parity error.
kStatus_UART_BaudrateNotSupport Baudrate is not support in current clock source.
kStatus_UART_IdleLineDetected UART IDLE line detected.
kStatus_UART_Timeout UART times out.

32.2.6.2 enum uart_parity_mode_t

Enumerator

kUART_ParityDisabled Parity disabled.
kUART_ParityEven Parity enabled, type even, bit setting: PE|PT = 10.
kUART_ParityOdd Parity enabled, type odd, bit setting: PE|PT = 11.

32.2.6.3 enum uart_stop_bit_count_t

Enumerator

kUART_OneStopBit One stop bit.
kUART_TwoStopBit Two stop bits.

32.2.6.4 enum uart_idle_type_select_t

Enumerator

kUART_IdleTypeStartBit Start counting after a valid start bit.
kUART_IdleTypeStopBit Start counting after a stop bit.

32.2.6.5 enum _uart_interrupt_enable

This structure contains the settings for all of the UART interrupt configurations.

Enumerator

kUART_LinBreakInterruptEnable LIN break detect interrupt.
kUART_RxActiveEdgeInterruptEnable RX active edge interrupt.

kUART_TxDataRegEmptyInterruptEnable Transmit data register empty interrupt.
kUART_TransmissionCompleteInterruptEnable Transmission complete interrupt.
kUART_RxDataRegFullInterruptEnable Receiver data register full interrupt.
kUART_IdleLineInterruptEnable Idle line interrupt.
kUART_RxOverrunInterruptEnable Receiver overrun interrupt.
kUART_NoiseErrorInterruptEnable Noise error flag interrupt.
kUART_FramingErrorInterruptEnable Framing error flag interrupt.
kUART_ParityErrorInterruptEnable Parity error flag interrupt.
kUART_RxFifoOverflowInterruptEnable RX FIFO overflow interrupt.
kUART_TxFifoOverflowInterruptEnable TX FIFO overflow interrupt.
kUART_RxFifoUnderflowInterruptEnable RX FIFO underflow interrupt.

32.2.6.6 anonymous enum

This provides constants for the UART status flags for use in the UART functions.

Enumerator

kUART_TxDataRegEmptyFlag TX data register empty flag.
kUART_TransmissionCompleteFlag Transmission complete flag.
kUART_RxDataRegFullFlag RX data register full flag.
kUART_IdleLineFlag Idle line detect flag.
kUART_RxOverrunFlag RX overrun flag.
kUART_NoiseErrorFlag RX takes 3 samples of each received bit. If any of these samples differ, noise flag sets
kUART_FramingErrorFlag Frame error flag, sets if logic 0 was detected where stop bit expected.
kUART_ParityErrorFlag If parity enabled, sets upon parity error detection.
kUART_LinBreakFlag LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled.
kUART_RxActiveEdgeFlag RX pin active edge interrupt flag, sets when active edge detected.
kUART_RxActiveFlag Receiver Active Flag (RAF), sets at beginning of valid start bit.
kUART_NoiseErrorInRxDataRegFlag Noisy bit, sets if noise detected.
kUART_ParityErrorInRxDataRegFlag Parity bit, sets if parity error detected.
kUART_TxFifoEmptyFlag TXEMPTY bit, sets if TX buffer is empty.
kUART_RxFifoEmptyFlag RXEMPTY bit, sets if RX buffer is empty.
kUART_TxFifoOverflowFlag TXOF bit, sets if TX buffer overflow occurred.
kUART_RxFifoOverflowFlag RXOF bit, sets if receive buffer overflow.
kUART_RxFifoUnderflowFlag RXUF bit, sets if receive buffer underflow.

32.2.7 Function Documentation

32.2.7.1 `uint32_t UARTGetInstance (UART_Type * base)`

Parameters

<i>base</i>	UART peripheral base address.
-------------	-------------------------------

Returns

UART instance.

32.2.7.2 status_t **UART_Init** (**UART_Type** * *base*, **const uart_config_t** * *config*, **uint32_t** *srcClock_Hz*)

This function configures the UART module with the user-defined settings. The user can configure the configuration structure and also get the default configuration by using the [UART_GetDefaultConfig\(\)](#) function. The example below shows how to use this API to configure UART.

```
* uart_config_t uartConfig;
* uartConfig.baudRate_Bps = 115200U;
* uartConfig.parityMode = kUART_ParityDisabled;
* uartConfig.stopBitCount = kUART_OneStopBit;
* uartConfig.txFifoWatermark = 0;
* uartConfig.rxFifoWatermark = 1;
* UART_Init(UART1, &uartConfig, 20000000U);
*
```

Parameters

<i>base</i>	UART peripheral base address.
<i>config</i>	Pointer to the user-defined configuration structure.
<i>srcClock_Hz</i>	UART clock source frequency in HZ.

Return values

<i>kStatus_UART_Baudrate-NotSupport</i>	Baudrate is not support in current clock source.
<i>kStatus_Success</i>	Status UART initialize succeed

32.2.7.3 void **UART_Deinit** (**UART_Type** * *base*)

This function waits for TX complete, disables TX and RX, and disables the UART clock.

Parameters

<i>base</i>	UART peripheral base address.
-------------	-------------------------------

32.2.7.4 void UART_GetDefaultConfig (*uart_config_t* * *config*)

This function initializes the UART configuration structure to a default value. The default values are as follows. *uartConfig->baudRate_Bps* = 115200U; *uartConfig->bitCountPerChar* = kUART_8BitsPerChar; *uartConfig->parityMode* = kUART_ParityDisabled; *uartConfig->stopBitCount* = kUART_OneStopBit; *uartConfig->txFifoWatermark* = 0; *uartConfig->rxFifoWatermark* = 1; *uartConfig->idleType* = kUART_IdleTypeStartBit; *uartConfig->enableTx* = false; *uartConfig->enableRx* = false;

Parameters

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

32.2.7.5 status_t UART_SetBaudRate (*UART_Type* * *base*, *uint32_t* *baudRate_Bps*, *uint32_t* *srcClock_Hz*)

This function configures the UART module baud rate. This function is used to update the UART module baud rate after the UART module is initialized by the *UART_Init*.

```
*   UART\_SetBaudRate(UART1, 115200U, 20000000U);
*
```

Parameters

<i>base</i>	UART peripheral base address.
<i>baudRate_Bps</i>	UART baudrate to be set.
<i>srcClock_Hz</i>	UART clock source frequency in Hz.

Return values

<i>kStatus_UART_Baudrate-NotSupport</i>	Baudrate is not support in the current clock source.
---	--

<i>kStatus_Success</i>	Set baudrate succeeded.
------------------------	-------------------------

32.2.7.6 void **UART_Enable9bitMode** (**UART_Type * base**, **bool enable**)

This function set the 9-bit mode for UART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

<i>base</i>	UART peripheral base address.
<i>enable</i>	true to enable, flase to disable.

32.2.7.7 static void **UART_SetMatchAddress** (**UART_Type * base**, **uint8_t address1**, **uint8_t address2**) [inline], [static]

This function configures the address for UART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receives with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note

Any UART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

<i>base</i>	UART peripheral base address.
<i>address1</i>	UART slave address 1.
<i>address2</i>	UART slave address 2.

32.2.7.8 static void **UART_EnableMatchAddress** (**UART_Type * base**, **bool match1**, **bool match2**) [inline], [static]

Parameters

<i>base</i>	UART peripheral base address.
<i>match1</i>	true to enable match address1, false to disable.
<i>match2</i>	true to enable match address2, false to disable.

32.2.7.9 static void UART_Set9thTransmitBit (**UART_Type** * *base*) [inline], [static]

Parameters

<i>base</i>	UART peripheral base address.
-------------	-------------------------------

32.2.7.10 static void UART_Clear9thTransmitBit (**UART_Type** * *base*) [inline], [static]

Parameters

<i>base</i>	UART peripheral base address.
-------------	-------------------------------

32.2.7.11 uint32_t UART_GetStatusFlags (**UART_Type** * *base*)

This function gets all UART status flags. The flags are returned as the logical OR value of the enumerators _uart_flags. To check a specific status, compare the return value with enumerators in _uart_flags. For example, to check whether the TX is empty, do the following.

```
*      if (kUART_TxDataRegEmptyFlag & UART_GetStatusFlags(UART1))
*
*      {
*          ...
*      }
```

Parameters

<i>base</i>	UART peripheral base address.
-------------	-------------------------------

Returns

UART status flags which are ORed by the enumerators in the _uart_flags.

32.2.7.12 status_t UART_ClearStatusFlags (**UART_Type** * *base*, **uint32_t** *mask*)

This function clears UART status flags with a provided mask. An automatically cleared flag can't be cleared by this function. These flags can only be cleared or set by hardware. kUART_TxDataRegEmptyFlag, kUART_TransmissionCompleteFlag, kUART_RxDataRegFullFlag, kUART_RxActiveFlag, kUART_NoiseErrorInRxDataRegFlag, kUART_ParityErrorInRxDataRegFlag, kUART_TxFifoEmptyFlag, kUART_RxFifoEmptyFlag

Note

that this API should be called when the Tx/Rx is idle. Otherwise it has no effect.

Parameters

<i>base</i>	UART peripheral base address.
<i>mask</i>	The status flags to be cleared; it is logical OR value of <code>_uart_flags</code> .

Return values

<i>kStatus_UART_FlagCannotClearManually</i>	The flag can't be cleared by this function but it is cleared automatically by hardware.
<i>kStatus_Success</i>	Status in the mask is cleared.

32.2.7.13 void UART_EnableInterrupts (**UART_Type** * *base*, **uint32_t** *mask*)

This function enables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [_uart_interrupt_enable](#). For example, to enable TX empty interrupt and RX full interrupt, do the following.

```
*     UART_EnableInterrupts(UART1,
                           kUART_TxDataRegEmptyInterruptEnable |
                           kUART_RxDataRegFullInterruptEnable);
*
```

Parameters

<i>base</i>	UART peripheral base address.
<i>mask</i>	The interrupts to enable. Logical OR of _uart_interrupt_enable .

32.2.7.14 void UART_DisableInterrupts (**UART_Type** * *base*, **uint32_t** *mask*)

This function disables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [_uart_interrupt_enable](#). For example, to disable TX empty interrupt and RX full interrupt do the following.

```
*     UART_DisableInterrupts(UART1,
*                           kUART_TxDataRegEmptyInterruptEnable |
*                           kUART_RxDataRegFullInterruptEnable);
*
```

Parameters

<i>base</i>	UART peripheral base address.
<i>mask</i>	The interrupts to disable. Logical OR of _uart_interrupt_enable .

32.2.7.15 uint32_t UART_GetEnabledInterrupts (**UART_Type * *base*)**

This function gets the enabled UART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators [_uart_interrupt_enable](#). To check a specific interrupts enable status, compare the return value with enumerators in [_uart_interrupt_enable](#). For example, to check whether TX empty interrupt is enabled, do the following.

```
*     uint32_t enabledInterrupts = UART_GetEnabledInterrupts(UART1);
*
*     if (kUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
*     {
*         ...
*     }
```

Parameters

<i>base</i>	UART peripheral base address.
-------------	-------------------------------

Returns

UART interrupt flags which are logical OR of the enumerators in [_uart_interrupt_enable](#).

32.2.7.16 static uint32_t UART_GetDataRegisterAddress (**UART_Type * *base*)
[**inline**], [**static**]**

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

<i>base</i>	UART peripheral base address.
-------------	-------------------------------

Returns

UART data register addresses which are used both by the transmitter and the receiver.

32.2.7.17 static void UART_EnableTxDMA (**UART_Type** * *base*, **bool** *enable*) [**inline**], [**static**]

This function enables or disables the transmit data register empty flag, S1[TDRE], to generate the DMA requests.

Parameters

<i>base</i>	UART peripheral base address.
<i>enable</i>	True to enable, false to disable.

32.2.7.18 static void UART_EnableRxDMA (**UART_Type** * *base*, **bool** *enable*) [**inline**], [**static**]

This function enables or disables the receiver data register full flag, S1[RDRF], to generate DMA requests.

Parameters

<i>base</i>	UART peripheral base address.
<i>enable</i>	True to enable, false to disable.

32.2.7.19 static void UART_EnableTx (**UART_Type** * *base*, **bool** *enable*) [**inline**], [**static**]

This function enables or disables the UART transmitter.

Parameters

<i>base</i>	UART peripheral base address.
<i>enable</i>	True to enable, false to disable.

32.2.7.20 static void UART_EnableRx (**UART_Type** * *base*, **bool** *enable*) [**inline**], [**static**]

This function enables or disables the UART receiver.

Parameters

<i>base</i>	UART peripheral base address.
<i>enable</i>	True to enable, false to disable.

32.2.7.21 static void UART_WriteByte (**UART_Type** * *base*, **uint8_t** *data*) [inline], [static]

This function writes data to the TX register directly. The upper layer must ensure that the TX register is empty or TX FIFO has empty room before calling this function.

Parameters

<i>base</i>	UART peripheral base address.
<i>data</i>	The byte to write.

32.2.7.22 static **uint8_t** UART_ReadByte (**UART_Type** * *base*) [inline], [static]

This function reads data from the RX register directly. The upper layer must ensure that the RX register is full or that the TX FIFO has data before calling this function.

Parameters

<i>base</i>	UART peripheral base address.
-------------	-------------------------------

Returns

The byte read from UART data register.

32.2.7.23 static **uint8_t** UART_GetRxFifoCount (**UART_Type** * *base*) [inline], [static]

Parameters

<i>base</i>	UART peripheral base address.
-------------	-------------------------------

Returns

rx FIFO data count.

32.2.7.24 **static uint8_t UART_GetTxFifoCount(UART_Type * *base*) [inline],
[static]**

Parameters

<i>base</i>	UART peripheral base address.
-------------	-------------------------------

Returns

tx FIFO data count.

32.2.7.25 void UART_SendAddress (**UART_Type** * *base*, **uint8_t** *address*)

Parameters

<i>base</i>	UART peripheral base address.
<i>address</i>	UART slave address.

32.2.7.26 status_t UART_WriteBlocking (**UART_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>	UART peripheral base address.
<i>data</i>	Start address of the data to write.
<i>length</i>	Size of the data to write.

Return values

<i>kStatus_UART_Timeout</i>	Transmission timed out and was aborted.
<i>kStatus_Success</i>	Successfully wrote all data.

32.2.7.27 status_t UART_ReadBlocking (**UART_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 reads data from the TX register.

Parameters

<i>base</i>	UART 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_UART_Rx-HardwareOverrun</i>	Receiver overrun occurred while receiving data.
<i>kStatus_UART_Noise-Error</i>	A noise error occurred while receiving data.
<i>kStatus_UART_Framing-Error</i>	A framing error occurred while receiving data.
<i>kStatus_UART_Parity-Error</i>	A parity error occurred while receiving data.
<i>kStatus_UART_Timeout</i>	Transmission timed out and was aborted.
<i>kStatus_Success</i>	Successfully received all data.

32.2.7.28 void UART_TransferCreateHandle (**UART_Type * base**, **uart_handle_t * handle**, **uart_transfer_callback_t callback**, **void * userData**)

This function initializes the UART handle which can be used for other UART transactional APIs. Usually, for a specified UART instance, call this API once to get the initialized handle.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>callback</i>	The callback function.
<i>userData</i>	The parameter of the callback function.

32.2.7.29 void UART_TransferStartRingBuffer (**UART_Type * base**, **uart_handle_t * handle**, **uint8_t * ringBuffer**, **size_t ringBufferSize**)

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received are stored into the ring buffer even when the user doesn't call the [UART_TransferReceiveNonBlocking\(\)](#) API. If data is already 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, only 31 bytes are used for saving data.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART 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.

32.2.7.30 void `UART_TransferStopRingBuffer (UART_Type * base, uart_handle_t * handle)`

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.

32.2.7.31 size_t `UART_TransferGetRxRingBufferLength (uart_handle_t * handle)`

Parameters

<i>handle</i>	UART handle pointer.
---------------	----------------------

Returns

Length of received data in RX ring buffer.

32.2.7.32 status_t `UART_TransferSendNonBlocking (UART_Type * base, uart_handle_t * handle, uart_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 ISR, the UART driver calls the callback function and passes the `kStatus_UART_TxIdle` as status parameter.

Note

The kStatus_UART_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out. Before disabling the TX, check the kUART_TransmissionCompleteFlag to ensure that the TX is finished.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>xfer</i>	UART transfer structure. See uart_transfer_t .

Return values

<i>kStatus_Success</i>	Successfully start the data transmission.
<i>kStatus_UART_TxBusy</i>	Previous transmission still not finished; data not all written to TX register yet.
<i>kStatus_InvalidArgument</i>	Invalid argument.

32.2.7.33 void UART_TransferAbortSend (**UART_Type * *base*, **uart_handle_t** * *handle*)**

This function aborts the interrupt-driven data sending. The user can get the remainBytes to find out how many bytes are not sent out.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.

32.2.7.34 status_t UART_TransferGetSendCount (**UART_Type * *base*, **uart_handle_t** * *handle*, **uint32_t** * *count*)**

This function gets the number of bytes sent out to bus by using the interrupt method.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>count</i>	Send bytes count.

Return values

<i>kStatus_NoTransferInProgress</i>	No send in progress.
<i>kStatus_InvalidArgument</i>	The parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <code>count</code> ;

32.2.7.35 `status_t UART_TransferReceiveNonBlocking (UART_Type * base, uart_handle_t * handle, uart_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 UART driver. When the new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter [k_Status_UART_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 UART 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.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>xfer</i>	UART transfer structure, see uart_transfer_t .
<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_UART_RxBusy</i>	Previous receive request is not finished.
<i>kStatus_InvalidArgument</i>	Invalid argument.

32.2.7.36 void UART_TransferAbortReceive (**UART_Type * *base*, **uart_handle_t** * *handle*)**

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to know how many bytes are not received yet.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.

32.2.7.37 status_t UART_TransferGetReceiveCount (**UART_Type** * *base*, **uart_handle_t** * *handle*, **uint32_t** * *count*)

This function gets the number of bytes that have been received.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART 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 <i>count</i> ;

32.2.7.38 status_t UART_EnableTxFIFO (**UART_Type** * *base*, **bool** *enable*)

This function enables or disables the UART Tx FIFO.

param *base* UART peripheral base address. param *enable* true to enable, false to disable. retval *kStatus_Success* Successfully turn on or turn off Tx FIFO. retval *kStatus_Fail* Fail to turn on or turn off Tx FIFO.

32.2.7.39 status_t UART_EnableRxFIFO (**UART_Type** * *base*, **bool** *enable*)

This function enables or disables the UART Rx FIFO.

param *base* UART peripheral base address. param *enable* true to enable, false to disable. retval *kStatus_Success* Successfully turn on or turn off Rx FIFO. retval *kStatus_Fail* Fail to turn on or turn off Rx FIFO.

32.2.7.40 static void UART_SetRxFifoWatermark (**UART_Type** * *base*, **uint8_t** *water*) [inline], [static]

Parameters

<i>base</i>	UART peripheral base address.
<i>water</i>	Rx FIFO watermark.

32.2.7.41 static void UART_SetTxFifoWatermark (**UART_Type** * *base*, **uint8_t** *water*) [**inline**], [**static**]

Parameters

<i>base</i>	UART peripheral base address.
<i>water</i>	Tx FIFO watermark.

32.2.7.42 void UART_TransferHandleIRQ (**UART_Type** * *base*, **void** * *irqHandle*)

This function handles the UART transmit and receive IRQ request.

Parameters

<i>base</i>	UART peripheral base address.
<i>irqHandle</i>	UART handle pointer.

32.2.7.43 void UART_TransferHandleErrorIRQ (**UART_Type** * *base*, **void** * *irqHandle*)

This function handles the UART error IRQ request.

Parameters

<i>base</i>	UART peripheral base address.
<i>irqHandle</i>	UART handle pointer.

32.2.8 Variable Documentation

32.2.8.1 **void*** *s_uartHandle*[]

32.2.8.2 **uart_isr_t** *s_uartIsr*

32.3 UART eDMA Driver

32.3.1 Overview

Data Structures

- struct [uart_edma_handle_t](#)
UART eDMA handle. [More...](#)

TypeDefs

- [typedef void\(* uart_edma_transfer_callback_t \)\(UART_Type *base, uart_edma_handle_t *handle, status_t status, void *userData\)](#)
UART transfer callback function.

Driver version

- [#define FSL_UART_EDMA_DRIVER_VERSION \(MAKE_VERSION\(2, 5, 2\)\)](#)
UART EDMA driver version.

eDMA transactional

- [void **UART_TransferCreateHandleEDMA** \(UART_Type *base, uart_edma_handle_t *handle, \[uart_edma_transfer_callback_t\]\(#\) callback, void *userData, \[edma_handle_t\]\(#\) *txEdmaHandle, \[edma_handle_t\]\(#\) *rxEdmaHandle\)](#)
Initializes the UART handle which is used in transactional functions.
- [status_t **UART_SendEDMA** \(UART_Type *base, uart_edma_handle_t *handle, \[uart_transfer_t\]\(#\) *xfer\)](#)
Sends data using eDMA.
- [status_t **UART_ReceiveEDMA** \(UART_Type *base, uart_edma_handle_t *handle, \[uart_transfer_t\]\(#\) *xfer\)](#)
Receives data using eDMA.
- [void **UART_TransferAbortSendEDMA** \(UART_Type *base, uart_edma_handle_t *handle\)](#)
Aborts the sent data using eDMA.
- [void **UART_TransferAbortReceiveEDMA** \(UART_Type *base, uart_edma_handle_t *handle\)](#)
Aborts the receive data using eDMA.
- [status_t **UART_TransferGetSendCountEDMA** \(UART_Type *base, uart_edma_handle_t *handle, uint32_t *count\)](#)
Gets the number of bytes that have been written to UART TX register.
- [status_t **UART_TransferGetReceiveCountEDMA** \(UART_Type *base, uart_edma_handle_t *handle, uint32_t *count\)](#)
Gets the number of received bytes.
- [void **UART_TransferEdmaHandleIRQ** \(UART_Type *base, void *uartEdmaHandle\)](#)
UART eDMA IRQ handle function.

32.3.2 Data Structure Documentation

32.3.2.1 struct _uart_edma_handle

Data Fields

- `uart_edma_transfer_callback_t callback`
Callback function.
- `void *userData`
UART callback function parameter.
- `size_t rxDataSizeAll`
Size of the data to receive.
- `size_t txDataSizeAll`
Size of the data to send out.
- `edma_handle_t *txEdmaHandle`
The eDMA TX channel used.
- `edma_handle_t *rxEdmaHandle`
The eDMA RX channel used.
- `uint8_t nbytes`
eDMA minor byte transfer count initially configured.
- `volatile uint8_t txState`
TX transfer state.
- `volatile uint8_t rxState`
RX transfer state.

Field Documentation

- (1) `uart_edma_transfer_callback_t uart_edma_handle_t::callback`
- (2) `void* uart_edma_handle_t::userData`
- (3) `size_t uart_edma_handle_t::rxDataSizeAll`
- (4) `size_t uart_edma_handle_t::txDataSizeAll`
- (5) `edma_handle_t* uart_edma_handle_t::txEdmaHandle`
- (6) `edma_handle_t* uart_edma_handle_t::rxEdmaHandle`
- (7) `uint8_t uart_edma_handle_t::nbytes`
- (8) `volatile uint8_t uart_edma_handle_t::txState`

32.3.3 Macro Definition Documentation

32.3.3.1 #define FSL_UART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 5, 2))

32.3.4 Typedef Documentation

32.3.4.1 `typedef void(* uart_edma_transfer_callback_t)(UART_Type *base, uart_edma_handle_t *handle, status_t status, void *userData)`

32.3.5 Function Documentation

32.3.5.1 `void UART_TransferCreateHandleEDMA (UART_Type * base, uart_edma_handle_t * handle, uart_edma_transfer_callback_t callback, void * userData, edma_handle_t * txEdmaHandle, edma_handle_t * rxEdmaHandle)`

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	Pointer to the <code>uart_edma_handle_t</code> structure.
<i>callback</i>	UART callback, NULL means no callback.
<i>userData</i>	User callback function data.
<i>rxEdmaHandle</i>	User-requested DMA handle for RX DMA transfer.
<i>txEdmaHandle</i>	User-requested DMA handle for TX DMA transfer.

32.3.5.2 `status_t UART_SendEDMA (UART_Type * base, uart_edma_handle_t * handle, uart_transfer_t * xfer)`

This function sends data using eDMA. 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>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>xfer</i>	UART eDMA transfer structure. See uart_transfer_t .

Return values

<i>kStatus_Success</i>	if succeeded; otherwise failed.
<i>kStatus_UART_TxBusy</i>	Previous transfer ongoing.
<i>kStatus_InvalidArgument</i>	Invalid argument.

32.3.5.3 status_t **UART_ReceiveEDMA** (**UART_Type** * *base*, **uart_edma_handle_t** * *handle*, **uart_transfer_t** * *xfer*)

This function receives data using eDMA. 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>	UART peripheral base address.
<i>handle</i>	Pointer to the <code>uart_edma_handle_t</code> structure.
<i>xfer</i>	UART eDMA transfer structure. See uart_transfer_t .

Return values

<i>kStatus_Success</i>	if succeeded; otherwise failed.
<i>kStatus_UART_RxBusy</i>	Previous transfer ongoing.
<i>kStatus_InvalidArgument</i>	Invalid argument.

32.3.5.4 void `UART_TransferAbortSendEDMA` (`UART_Type * base, uart_edma_handle_t * handle`)

This function aborts sent data using eDMA.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	Pointer to the <code>uart_edma_handle_t</code> structure.

32.3.5.5 void `UART_TransferAbortReceiveEDMA` (`UART_Type * base, uart_edma_handle_t * handle`)

This function aborts receive data using eDMA.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	Pointer to the <code>uart_edma_handle_t</code> structure.

32.3.5.6 status_t `UART_TransferGetSendCountEDMA` (`UART_Type * base, uart_edma_handle_t * handle, uint32_t * count`)

This function gets the number of bytes that have been written to UART TX register by DMA.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>count</i>	Send bytes count.

Return values

<i>kStatus_NoTransferIn-Progress</i>	No send in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <i>count</i> ;

32.3.5.7 **status_t UART_TransferGetReceiveCountEDMA (*UART_Type * base, uart_edma_handle_t * handle, uint32_t * count*)**

This function gets the number of received bytes.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>count</i>	Receive bytes count.

Return values

<i>kStatus_NoTransferIn-Progress</i>	No receive in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <i>count</i> ;

32.3.5.8 **void UART_TransferEdmaHandleIRQ (*UART_Type * base, void * uartEdmaHandle*)**

This function handles the UART transmit complete IRQ request and invoke user callback.

Parameters

<i>base</i>	UART peripheral base address.
<i>uartEdma-Handle</i>	UART handle pointer.

32.4 UART FreeRTOS Driver

32.4.1 Overview

Data Structures

- struct `uart_rtos_config_t`
UART configuration structure. [More...](#)

Driver version

- #define `FSL_UART_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 5, 0)`)
UART FreeRTOS driver version.

UART RTOS Operation

- int `UART_RTOS_Init` (`uart_rtos_handle_t *handle, uart_handle_t *t_handle, const uart_rtos_config_t *cfg`)
Initializes a UART instance for operation in RTOS.
- int `UART_RTOS_Deinit` (`uart_rtos_handle_t *handle`)
Deinitializes a UART instance for operation.

UART transactional Operation

- int `UART_RTOS_Send` (`uart_rtos_handle_t *handle, uint8_t *buffer, uint32_t length`)
Sends data in the background.
- int `UART_RTOS_Receive` (`uart_rtos_handle_t *handle, uint8_t *buffer, uint32_t length, size_t *received`)
Receives data.

32.4.2 Data Structure Documentation

32.4.2.1 struct `uart_rtos_config_t`

Data Fields

- `UART_Type * base`
UART base address.
- `uint32_t srclk`
UART source clock in Hz.
- `uint32_t baudrate`
Desired communication speed.
- `uart_parity_mode_t parity`
Parity setting.

- `uart_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.

32.4.3 Macro Definition Documentation

32.4.3.1 `#define FSL_UART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))`

32.4.4 Function Documentation

32.4.4.1 `int UART_RTOS_Init (uart_rtos_handle_t * handle, uart_handle_t * t_handle, const uart_rtos_config_t * cfg)`

Parameters

<code>handle</code>	The RTOS UART handle, the pointer to an allocated space for RTOS context.
<code>t_handle</code>	The pointer to the allocated space to store the transactional layer internal state.
<code>cfg</code>	The pointer to the parameters required to configure the UART after initialization.

Returns

0 succeed; otherwise fail.

32.4.4.2 `int UART_RTOS_Deinit (uart_rtos_handle_t * handle)`

This function deinitializes the UART module, sets all register values to reset value, and frees the resources.

Parameters

<code>handle</code>	The RTOS UART handle.
---------------------	-----------------------

32.4.4.3 `int UART_RTOS_Send (uart_rtos_handle_t * handle, 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 UART handle.
<i>buffer</i>	The pointer to the buffer to send.
<i>length</i>	The number of bytes to send.

32.4.4.4 int UART_RTOS_Receive (*uart_rtos_handle_t * handle*, *uint8_t * buffer*, *uint32_t length*, *size_t * received*)

This function receives data from UART. 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 UART handle.
<i>buffer</i>	The pointer to the buffer 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.

32.5 UART CMSIS Driver

This section describes the programming interface of the UART Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method see <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

The UART driver includes transactional APIs.

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 please write custom code.

32.5.1 UART CMSIS Driver

32.5.1.1 UART Send/receive using an interrupt method

```
/* UART callback */
void UART_Callback(uint32_t event)
{
    if (event == ARM_USART_EVENT_SEND_COMPLETE)
    {
        txBufferFull = false;
        txOnGoing = false;
    }

    if (event == ARM_USART_EVENT_RECEIVE_COMPLETE)
    {
        rxBufferEmpty = false;
        rxOnGoing = false;
    }
}

Driver_USART0.Initialize(UART_Callback);
Driver_USART0.PowerControl(ARM_POWER_FULL);
/* Send g_tipString out. */
txOnGoing = true;
Driver_USART0.Send(g_tipString, sizeof(g_tipString) - 1);

/* Wait send finished */
while (txOnGoing)
{
}
```

32.5.1.2 UART Send/Receive using the DMA method

```
/* UART callback */
void UART_Callback(uint32_t event)
{
    if (event == ARM_USART_EVENT_SEND_COMPLETE)
    {
        txBufferFull = false;
        txOnGoing = false;
    }

    if (event == ARM_USART_EVENT_RECEIVE_COMPLETE)
```

```
{  
    rxBUFFEREmpty = false;  
    rxOnGoing = false;  
}  
}  
  
Driver_USART0.Initialize(UART_Callback);  
DMAMGR_Init();  
Driver_USART0.PowerControl(ARM_POWER_FULL);  
  
/* Send g_tipString out. */  
txOnGoing = true;  
  
Driver_USART0.Send(g_tipString, sizeof(g_tipString) - 1);  
  
/* Wait send finished */  
while (txOnGoing)  
{  
}
```

Chapter 33

VREF: Voltage Reference Driver

33.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Crossbar Voltage Reference (VREF) block of MCUXpresso SDK devices.

The Voltage Reference(VREF) supplies an accurate 1.2 V voltage output that can be trimmed in 0.5 mV steps. VREF can be used in applications to provide a reference voltage to external devices and to internal analog peripherals, such as the ADC, DAC, or CMP. The voltage reference has operating modes that provide different levels of supply rejection and power consumption.

33.2 VREF functional Operation

To configure the VREF driver, configure `vref_config_t` structure in one of two ways.

1. Use the `VREF_GetDefaultConfig()` function.
2. Set the parameter in the `vref_config_t` structure.

To initialize the VREF driver, call the `VREF_Init()` function and pass a pointer to the `vref_config_t` structure.

To de-initialize the VREF driver, call the `VREF_Deinit()` function.

33.3 Typical use case and example

This example shows how to generate a reference voltage by using the VREF module.

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

Data Structures

- struct `vref_config_t`
The description structure for the VREF module. [More...](#)

Enumerations

- enum `vref_buffer_mode_t` {
 `kVREF_ModeBandgapOnly` = 0U,
 `kVREF_ModeHighPowerBuffer` = 1U,
 `kVREF_ModeLowPowerBuffer` = 2U }
VREF modes.

Driver version

- #define `FSL_VREF_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 2)`)
Version 2.1.2.

VREF functional operation

- void **VREF_Init** (VREF_Type *base, const vref_config_t *config)
Enables the clock gate and configures the VREF module according to the configuration structure.
- void **VREF_Deinit** (VREF_Type *base)
Stops and disables the clock for the VREF module.
- void **VREF_GetDefaultConfig** (vref_config_t *config)
Initializes the VREF configuration structure.
- void **VREF_SetTrimVal** (VREF_Type *base, uint8_t trimValue)
Sets a TRIM value for the reference voltage.
- static uint8_t **VREF_GetTrimVal** (VREF_Type *base)
Reads the value of the TRIM meaning output voltage.

33.4 Data Structure Documentation

33.4.1 struct vref_config_t

Data Fields

- vref_buffer_mode_t bufferMode
Buffer mode selection.

33.5 Macro Definition Documentation

33.5.1 #define FSL_VREF_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))

33.6 Enumeration Type Documentation

33.6.1 enum vref_buffer_mode_t

Enumerator

kVREF_ModeBandgapOnly Bandgap on only, for stabilization and startup.

kVREF_ModeHighPowerBuffer High-power buffer mode enabled.

kVREF_ModeLowPowerBuffer Low-power buffer mode enabled.

33.7 Function Documentation

33.7.1 void VREF_Init (VREF_Type * *base*, const vref_config_t * *config*)

This function must be called before calling all other VREF driver functions, read/write registers, and configurations with user-defined settings. The example below shows how to set up *vref_config_t* parameters and how to call the VREF_Init function by passing in these parameters. This is an example.

```
*     vref_config_t vrefConfig;
*     vrefConfig.bufferMode = kVREF_ModeHighPowerBuffer;
*     vrefConfig.enableExternalVoltRef = false;
*     vrefConfig.enableLowRef = false;
*     VREF_Init(VREF, &vrefConfig);
*
```

Parameters

<i>base</i>	VREF peripheral address.
<i>config</i>	Pointer to the configuration structure.

33.7.2 void VREF_Deinit (VREF_Type * *base*)

This function should be called to shut down the module. This is an example.

```
*     vref_config_t vrefUserConfig;
*     VREF_Init(VREF);
*     VREF_GetDefaultConfig(&vrefUserConfig);
*     ...
*     VREF_Deinit(VREF);
*
```

Parameters

<i>base</i>	VREF peripheral address.
-------------	--------------------------

33.7.3 void VREF_GetDefaultConfig (vref_config_t * *config*)

This function initializes the VREF configuration structure to default values. This is an example.

```
*     vrefConfig->bufferMode = kVREF_ModeHighPowerBuffer;
*     vrefConfig->enableExternalVoltRef = false;
*     vrefConfig->enableLowRef = false;
*
```

Parameters

<i>config</i>	Pointer to the initialization structure.
---------------	--

33.7.4 void VREF_SetTrimVal (VREF_Type * *base*, uint8_t *trimValue*)

This function sets a TRIM value for the reference voltage. Note that the TRIM value maximum is 0x3F.

Parameters

<i>base</i>	VREF peripheral address.
<i>trimValue</i>	Value of the trim register to set the output reference voltage (maximum 0x3F (6-bit)).

33.7.5 static uint8_t VREF_GetTrimVal (VREF_Type * *base*) [inline], [static]

This function gets the TRIM value from the TRM register.

Parameters

<i>base</i>	VREF peripheral address.
-------------	--------------------------

Returns

Six-bit value of trim setting.

Chapter 34

WDOG: Watchdog Timer Driver

34.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Watchdog module (WDOG) of MCUXpresso SDK devices.

34.2 Typical use case

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

Data Structures

- struct `wdog_work_mode_t`
Defines WDOG work mode. [More...](#)
- struct `wdog_config_t`
Describes WDOG configuration structure. [More...](#)
- struct `wdog_test_config_t`
Describes WDOG test mode configuration structure. [More...](#)

Enumerations

- enum `wdog_clock_source_t`{
 `kWDOG_LpoClockSource` = 0U,
 `kWDOG_AlternateClockSource` = 1U }
Describes WDOG clock source.
- enum `wdog_clock_prescaler_t`{
 `kWDOG_ClockPrescalerDivide1` = 0x0U,
 `kWDOG_ClockPrescalerDivide2` = 0x1U,
 `kWDOG_ClockPrescalerDivide3` = 0x2U,
 `kWDOG_ClockPrescalerDivide4` = 0x3U,
 `kWDOG_ClockPrescalerDivide5` = 0x4U,
 `kWDOG_ClockPrescalerDivide6` = 0x5U,
 `kWDOG_ClockPrescalerDivide7` = 0x6U,
 `kWDOG_ClockPrescalerDivide8` = 0x7U }
Describes the selection of the clock prescaler.
- enum `wdog_test_mode_t`{
 `kWDOG_QuickTest` = 0U,
 `kWDOG_ByeTest` = 1U }
Describes WDOG test mode.
- enum `wdog_tested_byte_t`{
 `kWDOG_TestByte0` = 0U,
 `kWDOG_TestByte1` = 1U,
 `kWDOG_TestByte2` = 2U,

```
kWDOG_TestByte3 = 3U }
```

Describes WDOG tested byte selection in byte test mode.

- enum `_wdog_interrupt_enable_t` { `kWDOG_InterruptEnable` = WDOG_STCTRLH_IRQRSTEN_-
MASK }
- WDOG interrupt configuration structure, default settings all disabled.*
- enum `_wdog_status_flags_t` {
 `kWDOG_RunningFlag` = WDOG_STCTRLH_WDOGEN_MASK,
 `kWDOG_TimeoutFlag` = WDOG_STCTRLL_INTFLG_MASK }
- WDOG status flags.*

Driver version

- #define `FSL_WDOG_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 1)`)
Defines WDOG driver version 2.0.1.

Unlock sequence

- #define `WDOG_FIRST_WORD_OF_UNLOCK` (0xC520U)
First word of unlock sequence.
- #define `WDOG_SECOND_WORD_OF_UNLOCK` (0xD928U)
Second word of unlock sequence.

Refresh sequence

- #define `WDOG_FIRST_WORD_OF_REFRESH` (0xA602U)
First word of refresh sequence.
- #define `WDOG_SECOND_WORD_OF_REFRESH` (0xB480U)
Second word of refresh sequence.

WDOG Initialization and De-initialization

- void `WDOG_GetDefaultConfig` (`wdog_config_t` *config)
Initializes the WDOG configuration structure.
- void `WDOG_Init` (`WDOG_Type` *base, const `wdog_config_t` *config)
Initializes the WDOG.
- void `WDOG_Deinit` (`WDOG_Type` *base)
Shuts down the WDOG.
- void `WDOG_SetTestModeConfig` (`WDOG_Type` *base, `wdog_test_config_t` *config)
Configures the WDOG functional test.

WDOG Functional Operation

- static void `WDOG_Enable` (`WDOG_Type` *base)
Enables the WDOG module.
- static void `WDOG_Disable` (`WDOG_Type` *base)
Disables the WDOG module.
- static void `WDOG_EnableInterrupts` (`WDOG_Type` *base, `uint32_t` mask)
Enables the WDOG interrupt.
- static void `WDOG_DisableInterrupts` (`WDOG_Type` *base, `uint32_t` mask)
Disables the WDOG interrupt.

- `uint32_t WDOG_GetStatusFlags (WDOG_Type *base)`
Gets the WDOG all status flags.
- `void WDOG_ClearStatusFlags (WDOG_Type *base, uint32_t mask)`
Clears the WDOG flag.
- `static void WDOG_SetTimeoutValue (WDOG_Type *base, uint32_t timeoutCount)`
Sets the WDOG timeout value.
- `static void WDOG_SetWindowValue (WDOG_Type *base, uint32_t windowValue)`
Sets the WDOG window value.
- `static void WDOG_Unlock (WDOG_Type *base)`
Unlocks the WDOG register written.
- `void WDOG_Refresh (WDOG_Type *base)`
Refreshes the WDOG timer.
- `static uint16_t WDOG_GetResetCount (WDOG_Type *base)`
Gets the WDOG reset count.
- `static void WDOG_ClearResetCount (WDOG_Type *base)`
Clears the WDOG reset count.

34.3 Data Structure Documentation

34.3.1 struct wdog_work_mode_t

Data Fields

- `bool enableWait`
Enables or disables WDOG in wait mode.
- `bool enableStop`
Enables or disables WDOG in stop mode.
- `bool enableDebug`
Enables or disables WDOG in debug mode.

34.3.2 struct wdog_config_t

Data Fields

- `bool enableWdog`
Enables or disables WDOG.
- `wdog_clock_source_t clockSource`
Clock source select.
- `wdog_clock_prescaler_t prescaler`
Clock prescaler value.
- `wdog_work_mode_t workMode`
Configures WDOG work mode in debug stop and wait mode.
- `bool enableUpdate`
Update write-once register enable.
- `bool enableInterrupt`
Enables or disables WDOG interrupt.
- `bool enableWindowMode`
Enables or disables WDOG window mode.

- `uint32_t windowValue`
Window value.
- `uint32_t timeoutValue`
Timeout value.

34.3.3 struct wdog_test_config_t

Data Fields

- `wdog_test_mode_t testMode`
Selects test mode.
- `wdog_tested_byte_t testedByte`
Selects tested byte in byte test mode.
- `uint32_t timeoutValue`
Timeout value.

34.4 Macro Definition Documentation

34.4.1 #define FSL_WDOG_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

34.5 Enumeration Type Documentation

34.5.1 enum wdog_clock_source_t

Enumerator

kWDOG_LpoClockSource WDOG clock sourced from LPO.

kWDOG_AlternateClockSource WDOG clock sourced from alternate clock source.

34.5.2 enum wdog_clock_prescaler_t

Enumerator

kWDOG_ClockPrescalerDivide1 Divided by 1.
kWDOG_ClockPrescalerDivide2 Divided by 2.
kWDOG_ClockPrescalerDivide3 Divided by 3.
kWDOG_ClockPrescalerDivide4 Divided by 4.
kWDOG_ClockPrescalerDivide5 Divided by 5.
kWDOG_ClockPrescalerDivide6 Divided by 6.
kWDOG_ClockPrescalerDivide7 Divided by 7.
kWDOG_ClockPrescalerDivide8 Divided by 8.

34.5.3 enum wdog_test_mode_t

Enumerator

kWDOG_QuickTest Selects quick test.

kWDOG_Bytetest Selects byte test.

34.5.4 enum wdog_tested_byte_t

Enumerator

kWDOG_TestByte0 Byte 0 selected in byte test mode.

kWDOG_TestByte1 Byte 1 selected in byte test mode.

kWDOG_TestByte2 Byte 2 selected in byte test mode.

kWDOG_TestByte3 Byte 3 selected in byte test mode.

34.5.5 enum _wdog_interrupt_enable_t

This structure contains the settings for all of the WDOG interrupt configurations.

Enumerator

kWDOG_InterruptEnable WDOG timeout generates an interrupt before reset.

34.5.6 enum _wdog_status_flags_t

This structure contains the WDOG status flags for use in the WDOG functions.

Enumerator

kWDOG_RunningFlag Running flag, set when WDOG is enabled.

kWDOG_TimeoutFlag Interrupt flag, set when an exception occurs.

34.6 Function Documentation

34.6.1 void WDOG_GetDefaultConfig (wdog_config_t * config)

This function initializes the WDOG configuration structure to default values. The default values are as follows.

```

*   wdogConfig->enableWdog = true;
*   wdogConfig->clockSource = kWDOG_IpoClockSource;
*   wdogConfig->prescaler = kWDOG_ClockPrescalerDivide1;
*   wdogConfig->workMode.enableWait = true;
*   wdogConfig->workMode.enableStop = false;
*   wdogConfig->workMode.enableDebug = false;
*   wdogConfig->enableUpdate = true;
*   wdogConfig->enableInterrupt = false;
*   wdogConfig->enableWindowMode = false;
*   wdogConfig->windowValue = 0;
*   wdogConfig->timeoutValue = 0xFFFFU;
*

```

Parameters

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

See Also

[wdog_config_t](#)

34.6.2 void WDOG_Init (WDOG_Type * *base*, const wdog_config_t * *config*)

This function initializes the WDOG. When called, the WDOG runs according to the configuration. To reconfigure WDOG without forcing a reset first, enableUpdate must be set to true in the configuration.

This is an example.

```

*   wdog_config_t config;
*   WDOG_GetDefaultConfig(&config);
*   config.timeoutValue = 0x7ffU;
*   config.enableUpdate = true;
*   WDOG_Init(wdog_base,&config);
*

```

Parameters

<i>base</i>	WDOG peripheral base address
<i>config</i>	The configuration of WDOG

34.6.3 void WDOG_Deinit (WDOG_Type * *base*)

This function shuts down the WDOG. Ensure that the WDOG_STCTRLH.ALLOWUPDATE is 1 which indicates that the register update is enabled.

34.6.4 void WDOG_SetTestModeConfig (WDOG_Type * *base*, wdog_test_config_t * *config*)

This function is used to configure the WDOG functional test. When called, the WDOG goes into test mode and runs according to the configuration. Ensure that the WDOG_STCTRLH.ALLOWUPDATE is 1 which means that the register update is enabled.

This is an example.

```
*     wdog_test_config_t test_config;
*     test_config.testMode = kWDOG_QuickTest;
*     test_config.timeoutValue = 0xfffffu;
*     WDOG_SetTestModeConfig(wdog_base, &test_config);
*
```

Parameters

<i>base</i>	WDOG peripheral base address
<i>config</i>	The functional test configuration of WDOG

34.6.5 static void WDOG_Enable (WDOG_Type * *base*) [inline], [static]

This function write value into WDOG_STCTRLH register to enable the WDOG, it is a write-once register, make sure that the WCT window is still open and this register has not been written in this WCT while this function is called.

Parameters

<i>base</i>	WDOG peripheral base address
-------------	------------------------------

34.6.6 static void WDOG_Disable (WDOG_Type * *base*) [inline], [static]

This function writes a value into the WDOG_STCTRLH register to disable the WDOG. It is a write-once register. Ensure that the WCT window is still open and that register has not been written to in this WCT while the function is called.

Parameters

<i>base</i>	WDOG peripheral base address
-------------	------------------------------

34.6.7 static void WDOG_EnableInterrupts (**WDOG_Type** * *base*, **uint32_t** *mask*) [inline], [static]

This function writes a value into the WDOG_STCTRLH register to enable the WDOG interrupt. It is a write-once register. Ensure that the WCT window is still open and the register has not been written to in this WCT while the function is called.

Parameters

<i>base</i>	WDOG peripheral base address
<i>mask</i>	The interrupts to enable The parameter can be combination of the following source if defined. <ul style="list-style-type: none"> • kWDOG_InterruptEnable

34.6.8 static void WDOG_DisableInterrupts (**WDOG_Type** * *base*, **uint32_t** *mask*) [inline], [static]

This function writes a value into the WDOG_STCTRLH register to disable the WDOG interrupt. It is a write-once register. Ensure that the WCT window is still open and the register has not been written to in this WCT while the function is called.

Parameters

<i>base</i>	WDOG peripheral base address
<i>mask</i>	The interrupts to disable The parameter can be combination of the following source if defined. <ul style="list-style-type: none"> • kWDOG_InterruptEnable

34.6.9 **uint32_t** WDOG_GetStatusFlags (**WDOG_Type** * *base*)

This function gets all status flags.

This is an example for getting the Running Flag.

```
*     uint32_t status;
*     status = WDOG_GetStatusFlags (wdog_base) &
*                           kWDOG_RunningFlag;
```

*

Parameters

<i>base</i>	WDOG peripheral base address
-------------	------------------------------

Returns

State of the status flag: asserted (true) or not-asserted (false).

See Also

[_wdog_status_flags_t](#)

- true: a related status flag has been set.
- false: a related status flag is not set.

34.6.10 void WDOG_ClearStatusFlags (WDOG_Type * *base*, uint32_t *mask*)

This function clears the WDOG status flag.

This is an example for clearing the timeout (interrupt) flag.

```
*   WDOG_ClearStatusFlags (wdog_base, kWDOG_TimeoutFlag);
```

Parameters

<i>base</i>	WDOG peripheral base address
<i>mask</i>	The status flags to clear. The parameter could be any combination of the following values. kWDOG_TimeoutFlag

34.6.11 static void WDOG_SetTimeoutValue (WDOG_Type * *base*, uint32_t *timeoutCount*) [inline], [static]

This function sets the timeout value. It should be ensured that the time-out value for the WDOG is always greater than 2xWCT time + 20 bus clock cycles. This function writes a value into WDOG_TOVALH and WDOG_TOVALL registers which are write-once. Ensure the WCT window is still open and the two registers have not been written to in this WCT while the function is called.

Parameters

<i>base</i>	WDOG peripheral base address
<i>timeoutCount</i>	WDOG timeout value; count of WDOG clock tick.

34.6.12 static void WDOG_SetWindowValue (WDOG_Type * *base*, uint32_t *windowValue*) [inline], [static]

This function sets the WDOG window value. This function writes a value into WDOG_WINH and WDOG_WINL registers which are write-once. Ensure the WCT window is still open and the two registers have not been written to in this WCT while the function is called.

Parameters

<i>base</i>	WDOG peripheral base address
<i>windowValue</i>	WDOG window value.

34.6.13 static void WDOG_Unlock (WDOG_Type * *base*) [inline], [static]

This function unlocks the WDOG register written. Before starting the unlock sequence and following configuration, disable the global interrupts. Otherwise, an interrupt may invalidate the unlocking sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

Parameters

<i>base</i>	WDOG peripheral base address
-------------	------------------------------

34.6.14 void WDOG_Refresh (WDOG_Type * *base*)

This function feeds the WDOG. This function should be called before the WDOG timer is in timeout. Otherwise, a reset is asserted.

Parameters

<i>base</i>	WDOG peripheral base address
-------------	------------------------------

**34.6.15 static uint16_t WDOG_GetResetCount(WDOG_Type * *base*) [inline],
[static]**

This function gets the WDOG reset count value.

Parameters

<i>base</i>	WDOG peripheral base address
-------------	------------------------------

Returns

WDOG reset count value.

34.6.16 static void WDOG_ClearResetCount(WDOG_Type * *base*) [inline], [static]

This function clears the WDOG reset count value.

Parameters

<i>base</i>	WDOG peripheral base address
-------------	------------------------------

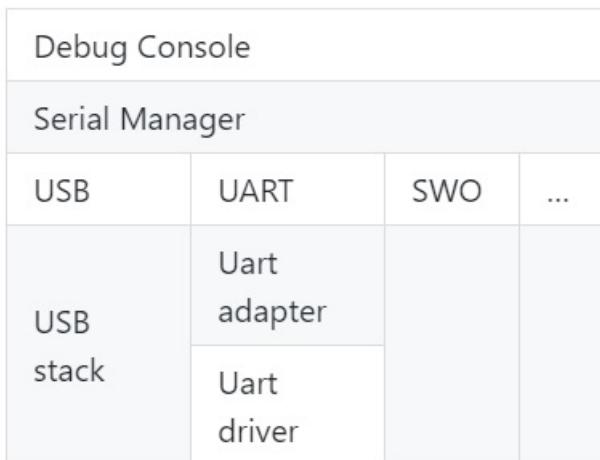
Chapter 35

Debug Console

35.1 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.



Debug console overview

35.2 Function groups

35.2.1 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,  
} 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);
```

35.2.2 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.

.precision	Description
.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.

length	Description
Do not support	

specifier	Description
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

- 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 *
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 == DEBUGCONSOLE_DISABLE /* Disable debug console */
#define PRINTF
#define SCANF
#define PUTCHAR
#define GETCHAR
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_SDK /* Select printf, scanf, putchar, getchar of SDK
```

```

version. */
#define PRINTF DbgConsole_Printf
#define SCANF DbgConsole_Scanf
#define PUTCHAR DbgConsole_Putchar
#define GETCHAR DbgConsole_Getchar
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN /* Select printf, scanf, putchar, getchar of
toolchain. */
#define PRINTF printf
#define SCANF scanf
#define PUTCHAR putchar
#define GETCHAR getchar
#endif /* SDK_DEBUGCONSOLE */

```

35.2.3 SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART

There are two macros `SDK_DEBUGCONSOLE` and `SDK_DEBUGCONSOLE_UART` added to configure `PRINTF` and low level output peripheral.

- The macro `SDK_DEBUGCONSOLE` is used for frontend. Whether debug console redirect to toolchain or SDK or disabled, it decides which is the frontend of the debug console, Tool chain or SDK. The function can be set by the macro `SDK_DEBUGCONSOLE`.
- The macro `SDK_DEBUGCONSOLE_UART` is used for backend. It is used to decide whether provide low level IO implementation to toolchain printf and scanf. For example, within MCUXpresso, if the macro `SDK_DEBUGCONSOLE_UART` is defined, `_sys_write` and `_sys_read` will be used when `_REDLIB_` is defined; `_write` and `_read` will be used in other cases. The macro does not specifically refer to the peripheral "UART". It refers to the external peripheral similar to UART, like as USB CDC, UART, SWO, etc. So if the macro `SDK_DEBUGCONSOLE_UART` is not defined when tool-chain printf is calling, the semihosting will be used.

The following matrix show the effects of `SDK_DEBUGCONSOLE` and `SDK_DEBUGCONSOLE_UART` on `PRINTF` and `printf`. The green mark is the default setting of the debug console.

<code>SDK_DEBUGCONSOLE</code>	<code>SDK_DEBUGCONSOLE_UART</code>	<code>PRINTF</code>	<code>printf</code>
<code>DEBUGCONSOLE_- REDIRECT_TO_SDK</code>	defined	Low level peripheral*	Low level peripheral
<code>DEBUGCONSOLE_- REDIRECT_TO_SDK</code>	undefined	Low level peripheral*	semihost
<code>DEBUGCONSOLE_- REDIRECT_TO_TO- OLCHAIN</code>	defined	Low level peripheral*	Low level peripheral
<code>DEBUGCONSOLE_- REDIRECT_TO_TO- OLCHAIN</code>	undefined	semihost	semihost
<code>DEBUGCONSOLE_- DISABLE</code>	defined	No output	Low level peripheral
<code>DEBUGCONSOLE_- DISABLE</code>	undefined	No output	semihost

- * the **low level peripheral** could be USB CDC, UART, or SWO, and so on.

35.3 Typical use case

Some examples use the PUTCHAR & GETCHAR function

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

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 equivalents 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 DEBUGCONSOLE_REDIRECT_TO_SDK`
Definition to select sdk or toolchain printf, 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.
- `status_t DbgConsole_EnterLowpower (void)`
Prepares to enter low power consumption.
- `status_t DbgConsole_ExitLowpower (void)`
Restores from low power consumption.
- `int DbgConsole_Printf (const char *fmt_s,...)`
Writes formatted output to the standard output stream.
- `int DbgConsole_Vprintf (const char *fmt_s, va_list formatStringArg)`
Writes formatted output to the standard output stream.
- `int DbgConsole_Putchar (int ch)`

- int [DbgConsole_Scanf](#) (char *fmt_s,...)
Writes a character to stdout.
- int [DbgConsole_Getchar](#) (void)
Reads formatted data from the standard input stream.
- int [DbgConsole_BlockingPrintf](#) (const char *fmt_s,...)
Reads a character from standard input.
- int [DbgConsole_BlockingVprintf](#) (const char *fmt_s, va_list formatStringArg)
Writes formatted output to the standard output stream with the blocking mode.
- int [DbgConsole_BlockingVfprintf](#) (const char *fmt_s, va_list formatStringArg)
Writes formatted output to the standard output stream with the blocking mode.
- status_t [DbgConsole_Flush](#) (void)
Debug console flush.

35.4 Macro Definition Documentation

35.4.1 #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U

Select toolchain printf and scanf.

35.4.2 #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U

35.4.3 #define DEBUGCONSOLE_DISABLE 2U

35.4.4 #define SDK_DEBUGCONSOLE DEBUGCONSOLE_REDIRECT_TO_SDK

The macro only support to be redefined in project setting.

35.4.5 #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.

35.5 Function Documentation

35.5.1 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. If the device is kSerialPort_Uart, the instance is UART peripheral instance. The UART hardware peripheral type is determined by UART adapter. For example, if the instance is 1, if the lpuart_adapter.c is added to the current project, the UART peripheral is LPUART1. If the uart_adapter.c is added to the current project, the UART peripheral is UART1.
<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"> • kSerialPort_Uart, • kSerialPort_UsbCdc
<i>clkSrcFreq</i>	Frequency of peripheral source clock.

Returns

Indicates whether initialization was successful or not.

Return values

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

35.5.2 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.

35.5.3 status_t DbgConsole_EnterLowpower (void)

This function is used to prepare to enter low power consumption.

Returns

Indicates whether de-initialization was successful or not.

35.5.4 status_t DbgConsole_ExitLowpower (void)

This function is used to restore from low power consumption.

Returns

Indicates whether de-initialization was successful or not.

35.5.5 int DbgConsole_Printf (const char * *fmt_s*, ...)

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

Parameters

<i>fmt_s</i>	Format control string.
--------------	------------------------

Returns

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

35.5.6 int DbgConsole_Vprintf (const char * *fmt_s*, va_list *formatStringArg*)

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

Parameters

<i>fmt_s</i>	Format control string.
<i>formatString-Arg</i>	Format arguments.

Returns

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

35.5.7 int DbgConsole_Putchar (int *ch*)

Call this function to write a character to stdout.

Parameters

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

Returns

Returns the character written.

35.5.8 int DbgConsole_Scanf (char * *fmt_s*, ...)

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_B-LOCKING 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>fmt_s</i>	Format control string.
--------------	------------------------

Returns

Returns the number of fields successfully converted and assigned.

35.5.9 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_B-LOCKING 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.

35.5.10 int DbgConsole_BlockingPrintf (const char * *fmt_s*, ...)

Call this function to write a formatted output to the standard output stream with the blocking mode. The function will send data with blocking mode no matter the DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set or not. The function could be used in system ISR mode with DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set.

Parameters

<i>fmt_s</i>	Format control string.
--------------	------------------------

Returns

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

35.5.11 int DbgConsole_BlockingVprintf (const char * *fmt_s*, va_list *formatStringArg*)

Call this function to write a formatted output to the standard output stream with the blocking mode. The function will send data with blocking mode no matter the DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set or not. The function could be used in system ISR mode with DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set.

Parameters

<i>fmt_s</i>	Format control string.
<i>formatString-Arg</i>	Format arguments.

Returns

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

35.5.12 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.

35.5.13 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 callbk function pointer

Returns

Number of characters to be print

35.5.14 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.

Return values

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

35.6 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.

35.6.1 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 `DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN`.

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. Please Make sure the `SDK_DEBUGCONSOLE_UART` is not defined in project settings.
4. Start the project by choosing Project>Download and Debug.
5. Choose View>Terminal I/O to display the output from the I/O operations.

35.6.2 Guide Semihosting for Keil µVision

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

35.6.3 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.

35.6.4 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

1. Download the image and set as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\twrk64f120m\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 = MK64FN1M0xxxx12
load semihosting_ARMGCC.elf
monitor reg pc = (0x00000004)
monitor reg sp = (0x00000000)
continue
```

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

35.7 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.

35.7.1 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, kSerialPort_Swo, clkSrcFreq);
```

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

Step 2: Building the project

Step 3: Download and run project

35.7.1.1 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.

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.

35.7.2 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 OK, 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.

35.7.3 Guide SWO for MCUXpresso IDE

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

35.7.4 Guide SWO for ARMGCC

NOTE: ARMGCC has no library support SWO.

Chapter 36

Notification Framework

36.1 Overview

This section describes the programming interface of the Notifier driver.

36.2 Notifier Overview

The Notifier provides a configuration dynamic change service. Based on this service, applications can switch between pre-defined configurations. The Notifier enables drivers and applications to register callback functions to this framework. Each time that the configuration is changed, drivers and applications receive a notification and change their settings. To simplify, the Notifier only supports the static callback registration. This means that, for applications, all callback functions are collected into a static table and passed to the Notifier.

These are the steps for the configuration transition.

1. Before configuration transition, the Notifier sends a "BEFORE" message to the callback table. When this message is received, IP drivers should check whether any current processes can be stopped and stop them. If the processes cannot be stopped, the callback function returns an error.
The Notifier supports two types of transition policies, a graceful policy and a forceful policy. When the graceful policy is used, if some callbacks return an error while sending a "BEFORE" message, the configuration transition stops and the Notifier sends a "RECOVER" message to all drivers that have stopped. Then, these drivers can recover the previous status and continue to work. When the forceful policy is used, drivers are stopped forcefully.
2. After the "BEFORE" message is processed successfully, the system switches to the new configuration.
3. After the configuration changes, the Notifier sends an "AFTER" message to the callback table to notify drivers that the configuration transition is finished.

This example shows how to use the Notifier in the Power Manager application.

```
#include "fsl_notifier.h"

// Definition of the Power Manager callback.
status_t callback0(notifier_notification_block_t *notify, void *data)
{
    status_t ret = kStatus_Success;

    ...
    ...

    return ret;
}
// Definition of the Power Manager user function.
status_t APP_PowerModeSwitch(notifier_user_config_t *targetConfig, void *
    userData)
```

```

{
    ...
    ...
    ...
}

...
...
...
...
...
...
// Main function.
int main(void)
{
    // Define a notifier handle.
    notifier_handle_t powerModeHandle;

    // Callback configuration.
    user_callback_data_t callbackData0;

    notifier_callback_config_t callbackCfg0 = {callback0,
        kNOTIFIER_CallbackBeforeAfter,
        (void *)&callbackData0};

    notifier_callback_config_t callbacks[] = {callbackCfg0};

    // Power mode configurations.
    power_user_config_t vlprConfig;
    power_user_config_t stopConfig;

    notifier_user_config_t *powerConfigs[] = {&vlprConfig, &stopConfig};

    // Definition of a transition to and out the power modes.
    vlprConfig.mode = kAPP_PowerModeVlpr;
    vlprConfig.enableLowPowerWakeUpOnInterrupt = false;

    stopConfig = vlprConfig;
    stopConfig.mode = kAPP_PowerModeStop;

    // Create Notifier handle.
    NOTIFIER_CreateHandle(&powerModeHandle, powerConfigs, 2U, callbacks, 1U,
        APP_PowerModeSwitch, NULL);
    ...

    ...
    // Power mode switch.
    NOTIFIER_switchConfig(&powerModeHandle, targetConfigIndex,
        kNOTIFIER_PolicyAgreement);
}

```

Data Structures

- struct [notifier_notification_block_t](#)
notification block passed to the registered callback function. [More...](#)
- struct [notifier_callback_config_t](#)
Callback configuration structure. [More...](#)
- struct [notifier_handle_t](#)
Notifier handle structure. [More...](#)

Typedefs

- [typedef void notifier_user_config_t](#)
Notifier user configuration type.
- [typedef status_t\(* notifier_user_function_t \)\(notifier_user_config_t *targetConfig, void *userData\)](#)

- *Notifier user function prototype Use this function to execute specific operations in configuration switch.*
typedef status_t(* notifier_callback_t)(notifier_notification_block_t *notify, void *data)
Callback prototype.

Enumerations

- **enum _notifier_status {**
kStatus_NOTIFIER_ErrorNotificationBefore,
kStatus_NOTIFIER_ErrorNotificationAfter }
Notifier error codes.
- **enum notifier_policy_t {**
kNOTIFIER_PolicyAgreement,
kNOTIFIER_PolicyForcible }
Notifier policies.
- **enum notifier_notification_type_t {**
kNOTIFIER_NotifyRecover = 0x00U,
kNOTIFIER_NotifyBefore = 0x01U,
kNOTIFIER_NotifyAfter = 0x02U }
Notification type.
- **enum notifier_callback_type_t {**
kNOTIFIER_CallbackBefore = 0x01U,
kNOTIFIER_CallbackAfter = 0x02U,
kNOTIFIER_CallbackBeforeAfter = 0x03U }
The callback type, which indicates kinds of notification the callback handles.

Functions

- **status_t NOTIFIER_CreateHandle (notifier_handle_t *notifierHandle, notifier_user_config_t **configs, uint8_t configsNumber, notifier_callback_config_t *callbacks, uint8_t callbacksNumber, notifier_user_function_t userFunction, void *userData)**
Creates a Notifier handle.
- **status_t NOTIFIER_SwitchConfig (notifier_handle_t *notifierHandle, uint8_t configIndex, notifier_policy_t policy)**
Switches the configuration according to a pre-defined structure.
- **uint8_t NOTIFIER_GetErrorCallbackIndex (notifier_handle_t *notifierHandle)**
This function returns the last failed notification callback.

36.3 Data Structure Documentation

36.3.1 struct notifier_notification_block_t

Data Fields

- **notifier_user_config_t * targetConfig**
Pointer to target configuration.
- **notifier_policy_t policy**
Configure transition policy.
- **notifier_notification_type_t notifyType**

Configure notification type.

Field Documentation

- (1) **notifier_user_config_t* notifier_notification_block_t::targetConfig**
- (2) **notifier_policy_t notifier_notification_block_t::policy**
- (3) **notifier_notification_type_t notifier_notification_block_t::notifyType**

36.3.2 struct notifier_callback_config_t

This structure holds the configuration of callbacks. Callbacks of this type are expected to be statically allocated. This structure contains the following application-defined data. callback - pointer to the callback function callbackType - specifies when the callback is called callbackData - pointer to the data passed to the callback.

Data Fields

- **notifier_callback_t callback**
Pointer to the callback function.
- **notifier_callback_type_t callbackType**
Callback type.
- **void * callbackData**
Pointer to the data passed to the callback.

Field Documentation

- (1) **notifier_callback_t notifier_callback_config_t::callback**
- (2) **notifier_callback_type_t notifier_callback_config_t::callbackType**
- (3) **void* notifier_callback_config_t::callbackData**

36.3.3 struct notifier_handle_t

Notifier handle structure. Contains data necessary for the Notifier proper function. Stores references to registered configurations, callbacks, information about their numbers, user function, user data, and other internal data. [NOTIFIER_CreateHandle\(\)](#) must be called to initialize this handle.

Data Fields

- **notifier_user_config_t ** configsTable**
Pointer to configure table.
- **uint8_t configsNumber**
Number of configurations.

- `notifier_callback_config_t * callbacksTable`
Pointer to callback table.
- `uint8_t callbacksNumber`
Maximum number of callback configurations.
- `uint8_t errorCallbackIndex`
Index of callback returns error.
- `uint8_t currentConfigIndex`
Index of current configuration.
- `notifier_user_function_t userFunction`
User function.
- `void * userData`
User data passed to user function.

Field Documentation

- (1) `notifier_user_config_t** notifier_handle_t::configsTable`
- (2) `uint8_t notifier_handle_t::configsNumber`
- (3) `notifier_callback_config_t* notifier_handle_t::callbacksTable`
- (4) `uint8_t notifier_handle_t::callbacksNumber`
- (5) `uint8_t notifier_handle_t::errorCallbackIndex`
- (6) `uint8_t notifier_handle_t::currentConfigIndex`
- (7) `notifier_user_function_t notifier_handle_t::userFunction`
- (8) `void* notifier_handle_t::userData`

36.4 Typedef Documentation

36.4.1 `typedef void notifier_user_config_t`

Reference of the user defined configuration is stored in an array; the notifier switches between these configurations based on this array.

36.4.2 `typedef status_t(* notifier_user_function_t)(notifier_user_config_t *targetConfig, void *userData)`

Before and after this function execution, different notification is sent to registered callbacks. If this function returns any error code, `NOTIFIER_SwitchConfig()` exits.

Parameters

<i>targetConfig</i>	target Configuration.
<i>userData</i>	Refers to other specific data passed to user function.

Returns

An error code or kStatus_Success.

36.4.3 **typedef status_t(* notifier_callback_t)(notifier_notification_block_t *notify, void *data)**

Declaration of a callback. It is common for registered callbacks. Reference to function of this type is part of the [notifier_callback_config_t](#) callback configuration structure. Depending on callback type, function of this prototype is called (see [NOTIFIER_SwitchConfig\(\)](#)) before configuration switch, after it or in both use cases to notify about the switch progress (see [notifier_callback_type_t](#)). When called, the type of the notification is passed as a parameter along with the reference to the target configuration structure (see [notifier_notification_block_t](#)) and any data passed during the callback registration. When notified before the configuration switch, depending on the configuration switch policy (see [notifier_policy_t](#)), the callback may deny the execution of the user function by returning an error code different than kStatus_Success (see [NOTIFIER_SwitchConfig\(\)](#)).

Parameters

<i>notify</i>	Notification block.
<i>data</i>	Callback data. Refers to the data passed during callback registration. Intended to pass any driver or application data such as internal state information.

Returns

An error code or kStatus_Success.

36.5 Enumeration Type Documentation

36.5.1 enum _notifier_status

Used as return value of Notifier functions.

Enumerator

kStatus_NOTIFIER_ErrorNotificationBefore An error occurs during send "BEFORE" notification.

kStatus_NOTIFIER_ErrorNotificationAfter An error occurs during send "AFTER" notification.

36.5.2 enum notifier_policy_t

Defines whether the user function execution is forced or not. For `kNOTIFIER_PolicyForcible`, the user function is executed regardless of the callback results, while `kNOTIFIER_PolicyAgreement` policy is used to exit `NOTIFIER_SwitchConfig()` when any of the callbacks returns error code. See also `NOTIFIER_SwitchConfig()` description.

Enumerator

kNOTIFIER_PolicyAgreement `NOTIFIER_SwitchConfig()` method is exited when any of the callbacks returns error code.

kNOTIFIER_PolicyForcible The user function is executed regardless of the results.

36.5.3 enum notifier_notification_type_t

Used to notify registered callbacks

Enumerator

kNOTIFIER_NotifyRecover Notify IP to recover to previous work state.

kNOTIFIER_NotifyBefore Notify IP that configuration setting is going to change.

kNOTIFIER_NotifyAfter Notify IP that configuration setting has been changed.

36.5.4 enum notifier_callback_type_t

Used in the callback configuration structure (`notifier_callback_config_t`) to specify when the registered callback is called during configuration switch initiated by the `NOTIFIER_SwitchConfig()`. Callback can be invoked in following situations.

- Before the configuration switch (Callback return value can affect `NOTIFIER_SwitchConfig()` execution. See the `NOTIFIER_SwitchConfig()` and `notifier_policy_t` documentation).
- After an unsuccessful attempt to switch configuration
- After a successful configuration switch

Enumerator

kNOTIFIER_CallbackBefore Callback handles BEFORE notification.

kNOTIFIER_CallbackAfter Callback handles AFTER notification.

kNOTIFIER_CallbackBeforeAfter Callback handles BEFORE and AFTER notification.

36.6 Function Documentation

36.6.1 **status_t NOTIFIER_CreateHandle (notifier_handle_t * *notifierHandle*,
notifier_user_config_t ** *configs*, uint8_t *configsNumber*, notifier_callback-
_config_t * *callbacks*, uint8_t *callbacksNumber*, notifier_user_function_t
userFunction, void * *userData*)**

Parameters

<i>notifierHandle</i>	A pointer to the notifier handle.
<i>configs</i>	A pointer to an array with references to all configurations which is handled by the Notifier.
<i>configsNumber</i>	Number of configurations. Size of the configuration array.
<i>callbacks</i>	A pointer to an array of callback configurations. If there are no callbacks to register during Notifier initialization, use NULL value.
<i>callbacks-Number</i>	Number of registered callbacks. Size of the callbacks array.
<i>userFunction</i>	User function.
<i>userData</i>	User data passed to user function.

Returns

An error Code or kStatus_Success.

36.6.2 status_t NOTIFIER_SwitchConfig (*notifier_handle_t * notifierHandle*, *uint8_t configIndex*, *notifier_policy_t policy*)

This function sets the system to the target configuration. Before transition, the Notifier sends notifications to all callbacks registered to the callback table. Callbacks are invoked in the following order: All registered callbacks are notified ordered by index in the callbacks array. The same order is used for before and after switch notifications. The notifications before the configuration switch can be used to obtain confirmation about the change from registered callbacks. If any registered callback denies the configuration change, further execution of this function depends on the notifier policy: the configuration change is either forced (kNOTIFIER_PolicyForcible) or exited (kNOTIFIER_PolicyAgreement). When configuration change is forced, the result of the before switch notifications are ignored. If an agreement is required, if any callback returns an error code, further notifications before switch notifications are cancelled and all already notified callbacks are re-invoked. The index of the callback which returned error code during pre-switch notifications is stored (any error codes during callbacks re-invocation are ignored) and NOTIFIER_GetErrorCallback() can be used to get it. Regardless of the policies, if any callback returns an error code, an error code indicating in which phase the error occurred is returned when NOTIFIER_SwitchConfig() exits.

Parameters

<i>notifierHandle</i>	pointer to notifier handle
<i>configIndex</i>	Index of the target configuration.
<i>policy</i>	Transaction policy, kNOTIFIER_PolicyAgreement or kNOTIFIER_PolicyForcible.

Returns

An error code or kStatus_Success.

36.6.3 uint8_t NOTIFIER_GetErrorCallbackIndex (*notifier_handle_t *notifierHandle*)

This function returns an index of the last callback that failed during the configuration switch while the last [NOTIFIER_SwitchConfig\(\)](#) was called. If the last [NOTIFIER_SwitchConfig\(\)](#) call ended successfully value equal to callbacks number is returned. The returned value represents an index in the array of static call-backs.

Parameters

<i>notifierHandle</i>	Pointer to the notifier handle
-----------------------	--------------------------------

Returns

Callback Index of the last failed callback or value equal to callbacks count.

Chapter 37

Shell

37.1 Overview

This section describes the programming interface of the Shell middleware.

Shell controls MCUs by commands via the specified communication peripheral based on the debug console driver.

37.2 Function groups

37.2.1 Initialization

To initialize the Shell middleware, call the `SHELL_Init()` function with these parameters. This function automatically enables the middleware.

```
shell_status_t SHELL_Init(shell_handle_t shellHandle,  
    serial_handle_t serialHandle, char *prompt);
```

Then, after the initialization was successful, call a command to control MCUs.

This example shows how to call the `SHELL_Init()` given the user configuration structure.

```
SHELL_Init(s_shellHandle, s_serialHandle, "Test@SHELL>");
```

37.2.2 Advanced Feature

- Support to get a character from standard input devices.

```
static shell_status_t SHELL_GetChar(shell_context_handle_t *shellContextHandle, uint8_t *ch);
```

Commands	Description
help	List all the registered commands.
exit	Exit program.

37.2.3 Shell Operation

```
SHELL_Init(s_shellHandle, s_serialHandle, "Test@SHELL>");  
SHELL_Task(s_shellHandle);
```

Data Structures

- struct `shell_command_t`
User command data configuration structure. More...

Macros

- #define `SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE`
Whether use non-blocking mode.
- #define `SHELL_AUTO_COMPLETE` (1U)
Macro to set on/off auto-complete feature.
- #define `SHELL_BUFFER_SIZE` (64U)
Macro to set console buffer size.
- #define `SHELL_MAX_ARGS` (8U)
Macro to set maximum arguments in command.
- #define `SHELL_HISTORY_COUNT` (3U)
Macro to set maximum count of history commands.
- #define `SHELL_IGNORE_PARAMETER_COUNT` (0xFF)
Macro to bypass arguments check.
- #define `SHELL_HANDLE_SIZE`
The handle size of the shell module.
- #define `SHELL_USE_COMMON_TASK` (0U)
Macro to determine whether use common task.
- #define `SHELL_TASK_PRIORITY` (2U)
Macro to set shell task priority.
- #define `SHELL_TASK_STACK_SIZE` (1000U)
Macro to set shell task stack size.
- #define `SHELL_HANDLE_DEFINE`(name) uint32_t name[((`SHELL_HANDLE_SIZE` + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
Defines the shell handle.
- #define `SHELL_COMMAND_DEFINE`(command, descriptor, callback, paramInt)
Defines the shell command structure.
- #define `SHELL_COMMAND`(command) &g_shellCommand##command
Gets the shell command pointer.

Typedefs

- typedef void * `shell_handle_t`
The handle of the shell module.
- typedef `shell_status_t`(* `cmd_function_t`)(`shell_handle_t` shellHandle, int32_t argc, char **argv)
User command function prototype.

Enumerations

- enum `shell_status_t` {

`kStatus_SHELL_Success` = kStatus_Success,

`kStatus_SHELL_Error` = MAKE_STATUS(kStatusGroup_SHELL, 1),

`kStatus_SHELL_OpenWriteHandleFailed` = MAKE_STATUS(kStatusGroup_SHELL, 2),

`kStatus_SHELL_OpenReadHandleFailed` = MAKE_STATUS(kStatusGroup_SHELL, 3) }

Shell status.

Shell functional operation

- `shell_status_t SHELL_Init (shell_handle_t shellHandle, serial_handle_t serialHandle, char *prompt)`
Initializes the shell module.
- `shell_status_t SHELL_RegisterCommand (shell_handle_t shellHandle, shell_command_t *shellCommand)`
Registers the shell command.
- `shell_status_t SHELL_UnregisterCommand (shell_command_t *shellCommand)`
Unregisters the shell command.
- `shell_status_t SHELL_Write (shell_handle_t shellHandle, const char *buffer, uint32_t length)`
Sends data to the shell output stream.
- `int SHELL_Printf (shell_handle_t shellHandle, const char *formatString,...)`
Writes formatted output to the shell output stream.
- `shell_status_t SHELL_WriteSynchronization (shell_handle_t shellHandle, const char *buffer, uint32_t length)`
Sends data to the shell output stream with OS synchronization.
- `int SHELL_PrintfSynchronization (shell_handle_t shellHandle, const char *formatString,...)`
Writes formatted output to the shell output stream with OS synchronization.
- `void SHELL_ChangePrompt (shell_handle_t shellHandle, char *prompt)`
Change shell prompt.
- `void SHELL_PrintPrompt (shell_handle_t shellHandle)`
Print shell prompt.
- `void SHELL_Task (shell_handle_t shellHandle)`
The task function for Shell.
- `static bool SHELL_checkRunningInIsr (void)`
Check if code is running in ISR.

37.3 Data Structure Documentation

37.3.1 struct shell_command_t

Data Fields

- `const char * pcCommand`
The command that is executed.
- `char * pcHelpString`
String that describes how to use the command.
- `const cmd_function_t pFuncCallBack`
A pointer to the callback function that returns the output generated by the command.
- `uint8_t cExpectedNumberOfParameters`
Commands expect a fixed number of parameters, which may be zero.
- `list_element_t link`
link of the element

Field Documentation

(1) `const char* shell_command_t::pcCommand`

For example "help". It must be all lower case.

(2) **char* shell_command_t::pcHelpString**

It should start with the command itself, and end with "\r\n". For example "help: Returns a list of all the commands\r\n".

(3) **const cmd_function_t shell_command_t::pFuncCallBack**(4) **uint8_t shell_command_t::cExpectedNumberOfParameters****37.4 Macro Definition Documentation****37.4.1 #define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE****37.4.2 #define SHELL_AUTO_COMPLETE (1U)****37.4.3 #define SHELL_BUFFER_SIZE (64U)****37.4.4 #define SHELL_MAX_ARGS (8U)****37.4.5 #define SHELL_HISTORY_COUNT (3U)****37.4.6 #define SHELL_HANDLE_SIZE****Value:**

```
(160U + SHELL_HISTORY_COUNT * SHELL_BUFFER_SIZE +
 SHELL_BUFFER_SIZE + SERIAL_MANAGER_READ_HANDLE_SIZE + \
 SERIAL_MANAGER_WRITE_HANDLE_SIZE)
```

It is the sum of the SHELL_HISTORY_COUNT * SHELL_BUFFER_SIZE + SHELL_BUFFER_SIZE + SERIAL_MANAGER_READ_HANDLE_SIZE + SERIAL_MANAGER_WRITE_HANDLE_SIZE

37.4.7 #define SHELL_USE_COMMON_TASK (0U)**37.4.8 #define SHELL_TASK_PRIORITY (2U)****37.4.9 #define SHELL_TASK_STACK_SIZE (1000U)**

37.4.10 #define SHELL_HANDLE_DEFINE(*name*) uint32_t *name*[(**SHELL_HANDLE_SIZE** + sizeof(uint32_t) - 1U) / sizeof(uint32_t)]

This macro is used to define a 4 byte aligned shell handle. Then use "(shell_handle_t)*name*" to get the shell handle.

The macro should be global and could be optional. You could also define shell handle by yourself.

This is an example,

```
* SHELL_HANDLE_DEFINE(shellHandle);
*
```

Parameters

<i>name</i>	The name string of the shell handle.
-------------	--------------------------------------

37.4.11 #define SHELL_COMMAND_DEFINE(*command*, *descriptor*, *callback*, *paramCount*)

Value:

```
\shell_command_t g_shellCommand##command = {
    (#command), (descriptor), (callback), (paramCount), {0},      \
}
```

This macro is used to define the shell command structure [shell_command_t](#). And then uses the macro SHELL_COMMAND to get the command structure pointer. The macro should not be used in any function.

This is a example,

```
* SHELL_COMMAND_DEFINE(exit, "\r\n\"exit\": Exit program\r\n", SHELL_ExitCommand, 0);
* SHELL_RegisterCommand(s_shellHandle, SHELL_COMMAND(exit));
*
```

Parameters

<i>command</i>	The command string of the command. The double quotes do not need. Such as exit for "exit", help for "Help", read for "read".
----------------	--

<i>descriptor</i>	The description of the command is used for showing the command usage when "help" is typing.
<i>callback</i>	The callback of the command is used to handle the command line when the input command is matched.
<i>paramCount</i>	The max parameter count of the current command.

37.4.12 #define SHELL_COMMAND(*command*) &g_shellCommand##*command*

This macro is used to get the shell command pointer. The macro should not be used before the macro SHELL_COMMAND_DEFINE is used.

Parameters

<i>command</i>	The command string of the command. The double quotes do not need. Such as exit for "exit", help for "Help", read for "read".
----------------	--

37.5 Typedef Documentation

37.5.1 typedef shell_status_t(* cmd_function_t)(shell_handle_t shellHandle, int32_t argc, char **argv)

37.6 Enumeration Type Documentation

37.6.1 enum shell_status_t

Enumerator

kStatus_SHELL_Success Success.

kStatus_SHELL_Error Failed.

kStatus_SHELL_OpenWriteHandleFailed Open write handle failed.

kStatus_SHELL_OpenReadHandleFailed Open read handle failed.

37.7 Function Documentation

37.7.1 shell_status_t SHELL_Init (shell_handle_t *shellHandle*, serial_handle_t *serialHandle*, char * *prompt*)

This function must be called before calling all other Shell functions. Call operation the Shell commands with user-defined settings. The example below shows how to set up the Shell and how to call the SHELL_Init function by passing in these parameters. This is an example.

```
* static SHELL_HANDLE_DEFINE(s_shellHandle);
* SHELL_Init((shell_handle_t)s_shellHandle,
*             (serial_handle_t)s_serialHandle, "Test@SHELL>");
*
```

Parameters

<i>shellHandle</i>	Pointer to point to a memory space of size SHELL_HANDLE_SIZE allocated by the caller. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SHELL_HANDLE_DEFINE(shellHandle) ; or <code>uint32_t shellHandle[((SHELL_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>
<i>serialHandle</i>	The serial manager module handle pointer.
<i>prompt</i>	The string prompt pointer of Shell. Only the global variable can be passed.

Return values

<i>kStatus_SHELL_Success</i>	The shell initialization succeed.
<i>kStatus_SHELL_Error</i>	An error occurred when the shell is initialized.
<i>kStatus_SHELL_OpenWriteHandleFailed</i>	Open the write handle failed.
<i>kStatus_SHELL_OpenReadHandleFailed</i>	Open the read handle failed.

37.7.2 **shell_status_t SHELL_RegisterCommand (shell_handle_t *shellHandle*, shell_command_t * *shellCommand*)**

This function is used to register the shell command by using the command configuration `shell_command_config_t`. This is a example,

```
* SHELL_COMMAND_DEFINE(exit, "\r\n\"exit\": Exit program\r\n", SHELL_ExitCommand, 0);
* SHELL_RegisterCommand(s_shellHandle, SHELL_COMMAND(exit));
*
```

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>shellCommand</i>	The command element.

Return values

<i>kStatus_SHELL_Success</i>	Successfully register the command.
<i>kStatus_SHELL_Error</i>	An error occurred.

37.7.3 shell_status_t SHELL_UnregisterCommand (shell_command_t * *shellCommand*)

This function is used to unregister the shell command.

Parameters

<i>shellCommand</i>	The command element.
---------------------	----------------------

Return values

<i>kStatus_SHELL_Success</i>	Successfully unregister the command.
------------------------------	--------------------------------------

37.7.4 shell_status_t SHELL_Write (shell_handle_t *shellHandle*, const char * *buffer*, uint32_t *length*)

This function is used to send data to the shell output stream.

Parameters

<i>shellHandle</i>	The shell 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_SHELL_Success</i>	Successfully send data.
<i>kStatus_SHELL_Error</i>	An error occurred.

37.7.5 int SHELL_Printf (shell_handle_t *shellHandle*, const char * *formatString*, ...)

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

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>formatString</i>	Format string.

Returns

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

37.7.6 **shell_status_t SHELL_WriteSynchronization (shell_handle_t *shellHandle*, const char * *buffer*, uint32_t *length*)**

This function is used to send data to the shell output stream with OS synchronization, note the function could not be called in ISR.

Parameters

<i>shellHandle</i>	The shell 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_SHELL_Success</i>	Successfully send data.
<i>kStatus_SHELL_Error</i>	An error occurred.

37.7.7 **int SHELL_PrintfSynchronization (shell_handle_t *shellHandle*, const char * *formatString*, ...)**

Call this function to write a formatted output to the shell output stream with OS synchronization, note the function could not be called in ISR.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
--------------------	----------------------------------

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

Returns

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

37.7.8 void SHELL_ChangePrompt (shell_handle_t *shellHandle*, char * *prompt*)

Call this function to change shell prompt.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>prompt</i>	The string which will be used for command prompt

Returns

NULL.

37.7.9 void SHELL_PrintPrompt (shell_handle_t *shellHandle*)

Call this function to print shell prompt.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
--------------------	----------------------------------

Returns

NULL.

37.7.10 void SHELL_Task (shell_handle_t *shellHandle*)

The task function for Shell; The function should be polled by upper layer. This function does not return until Shell command exit was called.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
--------------------	----------------------------------

37.7.11 static bool SHELL_checkRunningInIsr(void) [inline], [static]

This function is used to check if code running in ISR.

Return values

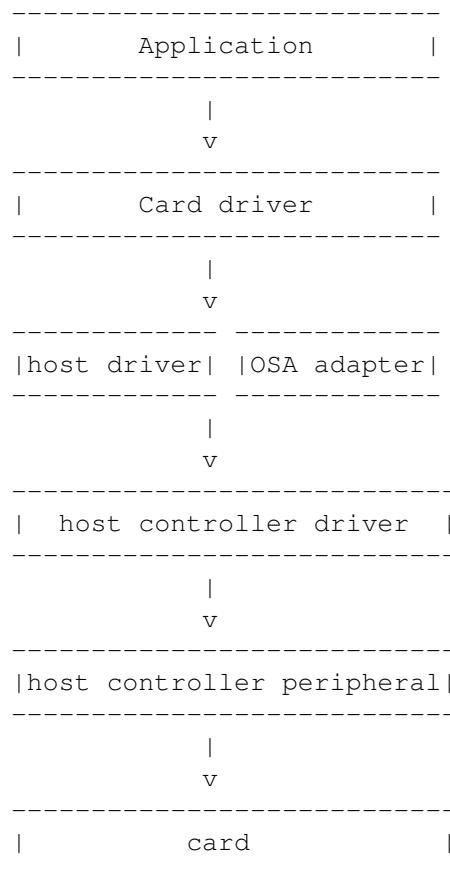
<i>TRUE</i>	if code runing in ISR.
-------------	------------------------

Chapter 38

Cards: Secure Digital Card/Embedded MultiMedia Card/SD-IO Card

38.1 Overview

The MCUXpresso SDK provides drivers to access the Secure Digital Card(up to v3.0), Embedded Multi-Media Card(up to v5.0) and sdio card(up to v3.0) based on the SDHC/USDHC/SDIF driver. Here is a simple block diagram about the drivers:



Modules

- [SDMMC Common](#)
- [SDMMC HOST Driver](#)

38.2 SDMMC HOST Driver

The host adapter driver provide adapter for blocking/non_blocking mode.

38.3 SDMMC Common

38.3.1 Overview

The sdmmc common function and definition.

Data Structures

- struct `sd_detect_card_t`
`sd card detect` [More...](#)
- struct `sd_io_voltage_t`
`io voltage control configuration` [More...](#)
- struct `sd_usr_param_t`
`sdcard user parameter` [More...](#)
- struct `sdio_card_int_t`
`card interrupt application callback` [More...](#)
- struct `sdio_usr_param_t`
`sdio user parameter` [More...](#)
- struct `sdio_fbr_t`
`sdio card FBR register` [More...](#)
- struct `sdio_common_cis_t`
`sdio card common CIS` [More...](#)
- struct `sdio_func_cis_t`
`sdio card function CIS` [More...](#)
- struct `sd_status_t`
`SD card status.` [More...](#)
- struct `sd_cid_t`
`SD card CID register.` [More...](#)
- struct `sd_csd_t`
`SD card CSD register.` [More...](#)
- struct `sd_scr_t`
`SD card SCR register.` [More...](#)
- struct `mmc_cid_t`
`MMC card CID register.` [More...](#)
- struct `mmc_csd_t`
`MMC card CSD register.` [More...](#)
- struct `mmc_extended_csd_t`
`MMC card Extended CSD register (unit: byte).` [More...](#)
- struct `mmc_extended_csd_config_t`
`MMC Extended CSD configuration.` [More...](#)
- struct `mmc_boot_config_t`
`MMC card boot configuration definition.` [More...](#)

Macros

- #define `SWAP_WORD_BYTSEQUENCE(x)` (`__REV(x)`)
`Reverse byte sequence in uint32_t.`
- #define `SWAP_HALF_WROD_BYTSEQUENCE(x)` (`__REV16(x)`)

- Reverse byte sequence for each half word in `uint32_t`.
- `#define FSL_SDMMC_MAX_VOLTAGE_RETRIES` (1000U)
Maximum loop count to check the card operation voltage range.
- `#define FSL_SDMMC_MAX_CMD_RETRIES` (10U)
Maximum loop count to send the cmd.
- `#define FSL_SDMMC_DEFAULT_BLOCK_SIZE` (512U)
Default block size.
- `#define SDMMC_DATA_BUFFER_ALIGN_CACHE` `sizeof(uint32_t)`
make sure the internal buffer address is cache align
- `#define FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE` (`FSL_SDMMC_DEFAULT_BLOCK_SIZE + SDMMC_DATA_BUFFER_ALIGN_CACHE`)
`sdmmc card internal buffer size`
- `#define FSL_SDMMC_CARD_MAX_BUS_FREQ`(max, target) ((max) == 0U ? (target) : ((max) > (target) ? (target) : (max)))
`get maximum freq`
- `#define SDMMC_LOG`(format,...)
`SD/MMC error log.`
- `#define SDMMC_CLOCK_400KHZ` (400000U)
`SD/MMC card initialization clock frequency.`
- `#define SD_CLOCK_25MHZ` (25000000U)
`SD card bus frequency 1 in high-speed mode.`
- `#define SD_CLOCK_50MHZ` (50000000U)
`SD card bus frequency 2 in high-speed mode.`
- `#define SD_CLOCK_100MHZ` (100000000U)
`SD card bus frequency in SDR50 mode.`
- `#define SD_CLOCK_208MHZ` (208000000U)
`SD card bus frequency in SDR104 mode.`
- `#define MMC_CLOCK_26MHZ` (26000000U)
`MMC card bus frequency 1 in high-speed mode.`
- `#define MMC_CLOCK_52MHZ` (52000000U)
`MMC card bus frequency 2 in high-speed mode.`
- `#define MMC_CLOCK_DDR52` (52000000U)
`MMC card bus frequency in high-speed DDR52 mode.`
- `#define MMC_CLOCK_HS200` (200000000U)
`MMC card bus frequency in high-speed HS200 mode.`
- `#define MMC_CLOCK_HS400` (400000000U)
`MMC card bus frequency in high-speed HS400 mode.`
- `#define SDMMC_MASK`(bit) (`1UL << (bit)`)
`mask convert`
- `#define SDMMC_R1_ALL_ERROR_FLAG`
`R1 all the error flag.`
- `#define SDMMC_R1_CURRENT_STATE`(x) (((x)&0x00001E00U) >> 9U)
`R1: current state.`
- `#define SDSPI_R7_VERSION_SHIFT` (28U)
`The bit mask for COMMAND VERSION field in R7.`
- `#define SDSPI_R7_VERSION_MASK` (0xFU)
`The bit mask for COMMAND VERSION field in R7.`
- `#define SDSPI_R7_VOLTAGE_SHIFT` (8U)
`The bit shift for VOLTAGE ACCEPTED field in R7.`
- `#define SDSPI_R7_VOLTAGE_MASK` (0xFU)
`The bit mask for VOLTAGE ACCEPTED field in R7.`

- #define **SDSPI_R7_VOLTAGE_27_36_MASK** (0x1U << SDSPI_R7_VOLTAGE_SHIFT)

The bit mask for VOLTAGE 2.7V to 3.6V field in R7.
- #define **SDSPI_R7_ECHO_SHIFT** (0U)

The bit shift for ECHO field in R7.
- #define **SDSPI_R7_ECHO_MASK** (0xFFU)

The bit mask for ECHO field in R7.
- #define **SDSPI_DATA_ERROR_TOKEN_MASK** (0xFU)

Data error token mask.
- #define **SDSPI_DATA_RESPONSE_TOKEN_MASK** (0x1FU)

Mask for data response bits.
- #define **SDIO_CCCR_REG_NUMBER** (0x16U)

sdio card cccr register number
- #define **SDIO_IO_READY_TIMEOUT_UNIT** (10U)

sdio IO ready timeout steps
- #define **SDIO_CMD_ARGUMENT_RW_POS** (31U)

read/write flag position
- #define **SDIO_CMD_ARGUMENT_FUNC_NUM_POS** (28U)

function number position
- #define **SDIO_DIRECT_CMD_ARGUMENT_RAW_POS** (27U)

direct raw flag position
- #define **SDIO_CMD_ARGUMENT_REG_ADDR_POS** (9U)

direct reg addr position
- #define **SDIO_CMD_ARGUMENT_REG_ADDR_MASK** (0x1FFFFU)

direct reg addr mask
- #define **SDIO_DIRECT_CMD_DATA_MASK** (0xFFU)

data mask
- #define **SDIO_EXTEND_CMD_ARGUMENT_BLOCK_MODE_POS** (27U)

extended command argument block mode bit position
- #define **SDIO_EXTEND_CMD_ARGUMENT_OP_CODE_POS** (26U)

extended command argument OP Code bit position
- #define **SDIO_EXTEND_CMD_BLOCK_MODE_MASK** (0x08000000U)

block mode mask
- #define **SDIO_EXTEND_CMD_OP_CODE_MASK** (0x04000000U)

op code mask
- #define **SDIO_EXTEND_CMD_COUNT_MASK** (0x1FFU)

byte/block count mask
- #define **SDIO_MAX_BLOCK_SIZE** (2048U)

max block size
- #define **SDIO_FBR_BASE**(x) ((x)*0x100U)

function basic register
- #define **SDIO_TPL_CODE_END** (0xFFU)

tuple end
- #define **SDIO_TPL_CODE_MANIFID** (0x20U)

manufacturer ID
- #define **SDIO_TPL_CODE_FUNCID** (0x21U)

function ID
- #define **SDIO_TPL_CODE_FUNCE** (0x22U)

function extension tuple
- #define **SDIO_OCR_VOLTAGE_WINDOW_MASK** (0xFFFFU << 8U)

sdio ocr voltage window mask
- #define **SDIO_OCR_IO_NUM_MASK** (7U << kSDIO_OcrIONumber)

- `sdio ocr reigster IO NUMBER mask`
- `#define SDIO_CCCR_SUPPORT_HIGHSPEED (1UL << 9U)`
UHS timing mode flag.
- `#define SDIO_CCCR_DRIVER_TYPE_MASK (3U << 4U)`
Driver type flag.
- `#define SDIO_CCCR_ASYNC_INT_MASK (1U)`
async interrupt flag
- `#define SDIO_CCCR_SUPPORT_8BIT_BUS (1UL << 18U)`
8 bit data bus flag
- `#define MMC_OCR_V170TO195_SHIFT (7U)`
The bit mask for VOLTAGE WINDOW 1.70V to 1.95V field in OCR.
- `#define MMC_OCR_V170TO195_MASK (0x00000080U)`
The bit mask for VOLTAGE WINDOW 1.70V to 1.95V field in OCR.
- `#define MMC_OCR_V200TO260_SHIFT (8U)`
The bit shift for VOLTAGE WINDOW 2.00V to 2.60V field in OCR.
- `#define MMC_OCR_V200TO260_MASK (0x00007F00U)`
The bit mask for VOLTAGE WINDOW 2.00V to 2.60V field in OCR.
- `#define MMC_OCR_V270TO360_SHIFT (15U)`
The bit shift for VOLTAGE WINDOW 2.70V to 3.60V field in OCR.
- `#define MMC_OCR_V270TO360_MASK (0x00FF8000U)`
The bit mask for VOLTAGE WINDOW 2.70V to 3.60V field in OCR.
- `#define MMC_OCR_ACCESS_MODE_SHIFT (29U)`
The bit shift for ACCESS MODE field in OCR.
- `#define MMC_OCR_ACCESS_MODE_MASK (0x60000000U)`
The bit mask for ACCESS MODE field in OCR.
- `#define MMC_OCR_BUSY_SHIFT (31U)`
The bit shift for BUSY field in OCR.
- `#define MMC_OCR_BUSY_MASK (1U << MMC_OCR_BUSY_SHIFT)`
The bit mask for BUSY field in OCR.
- `#define MMC_TRANSFER_SPEED_FREQUENCY_UNIT_SHIFT (0U)`
The bit shift for FREQUENCY UNIT field in TRANSFER SPEED(TRAN-SPEED in Extended CSD)
- `#define MMC_TRANSFER_SPEED_FREQUENCY_UNIT_MASK (0x07U)`
The bit mask for FRQEUNCY UNIT in TRANSFER SPEED.
- `#define MMC_TRANSFER_SPEED_MULTIPLIER_SHIFT (3U)`
The bit shift for MULTIPLIER field in TRANSFER SPEED.
- `#define MMC_TRANSFER_SPEED_MULTIPLIER_MASK (0x78U)`
The bit mask for MULTIPLIER field in TRANSFER SPEED.
- `#define READ_MMC_TRANSFER_SPEED_FREQUENCY_UNIT(CSD) (((CSD).transferSpeed) & MMC_TRANSFER_SPEED_FREQUENCY_UNIT_MASK) >> MMC_TRANSFER_SPEED_FREQUENCY_UNIT_SHIFT)`
Read the value of FREQUENCY UNIT in TRANSFER SPEED.
- `#define READ_MMC_TRANSFER_SPEED_MULTIPLIER(CSD) (((CSD).transferSpeed) & MMC_TRANSFER_SPEED_MULTIPLIER_MASK) >> MMC_TRANSFER_SPEED_MULTIPLIER_SHIFT)`
Read the value of MULTIPLIER filed in TRANSFER SPEED.
- `#define MMC_POWER_CLASS_4BIT_MASK (0x0FU)`
The power class value bit mask when bus in 4 bit mode.
- `#define MMC_POWER_CLASS_8BIT_MASK (0xF0U)`
The power class current value bit mask when bus in 8 bit mode.
- `#define MMC_CACHE_CONTROL_ENABLE (1U)`
mmc cache control enable

- #define **MMC_CACHE_TRIGGER_FLUSH** (1U)
mmc cache flush
- #define **MMC_DATA_BUS_WIDTH_TYPE_NUMBER** (3U)
The number of data bus width type.
- #define **MMC_PARTITION_CONFIG_PARTITION_ACCESS_SHIFT** (0U)
The bit shift for PARTITION ACCESS filed in BOOT CONFIG (BOOT_CONFIG in Extend CSD)
- #define **MMC_PARTITION_CONFIG_PARTITION_ACCESS_MASK** (0x00000007U)
The bit mask for PARTITION ACCESS field in BOOT CONFIG.
- #define **MMC_PARTITION_CONFIG_PARTITION_ENABLE_SHIFT** (3U)
The bit shift for PARTITION ENABLE field in BOOT CONFIG.
- #define **MMC_PARTITION_CONFIG_PARTITION_ENABLE_MASK** (0x00000038U)
The bit mask for PARTITION ENABLE field in BOOT CONFIG.
- #define **MMC_PARTITION_CONFIG_BOOT_ACK_SHIFT** (6U)
The bit shift for ACK field in BOOT CONFIG.
- #define **MMC_PARTITION_CONFIG_BOOT_ACK_MASK** (0x00000040U)
The bit mask for ACK field in BOOT CONFIG.
- #define **MMC_BOOT_BUS_CONDITION_BUS_WIDTH_SHIFT** (0U)
The bit shift for BOOT BUS WIDTH field in BOOT CONFIG.
- #define **MMC_BOOT_BUS_CONDITION_BUS_WIDTH_MASK** (3U)
The bit mask for BOOT BUS WIDTH field in BOOT CONFIG.
- #define **MMC_BOOT_BUS_CONDITION_RESET_BUS_CONDITION_SHIFT** (2U)
The bit shift for BOOT BUS WIDTH RESET field in BOOT CONFIG.
- #define **MMC_BOOT_BUS_CONDITION_RESET_BUS_CONDITION_MASK** (4U)
The bit mask for BOOT BUS WIDTH RESET field in BOOT CONFIG.
- #define **MMC_BOOT_BUS_CONDITION_BOOT_MODE_SHIFT** (3U)
The bit shift for BOOT MODE field in BOOT CONFIG.
- #define **MMC_BOOT_BUS_CONDITION_BOOT_MODE_MASK** (0x18U)
The bit mask for BOOT MODE field in BOOT CONFIG.
- #define **MMC_EXTENDED_CSD_BYTES** (512U)
The length of Extended CSD register, unit as bytes.
- #define **MMC_DEFAULT_RELATIVE_ADDRESS** (2UL)
MMC card default relative address.
- #define **SD_PRODUCT_NAME_BYTES** (5U)
SD card product name length united as bytes.
- #define **SD_AU_START_VALUE** (1U)
SD AU start value.
- #define **SD_UHS_AU_START_VALUE** (7U)
SD UHS AU start value.
- #define **SD_TRANSFER_SPEED_RATE_UNIT_SHIFT** (0U)
The bit shift for RATE UNIT field in TRANSFER SPEED.
- #define **SD_TRANSFER_SPEED_RATE_UNIT_MASK** (0x07U)
The bit mask for RATE UNIT field in TRANSFER SPEED.
- #define **SD_TRANSFER_SPEED_TIME_VALUE_SHIFT** (2U)
The bit shift for TIME VALUE field in TRANSFER SPEED.
- #define **SD_TRANSFER_SPEED_TIME_VALUE_MASK** (0x78U)
The bit mask for TIME VALUE field in TRANSFER SPEED.
- #define **SD_RD_TRANSFER_SPEED_RATE_UNIT(x)** (((x.transferSpeed) & **SD_TRANSFER_SPEED_RATE_UNIT_MASK**) >> **SD_TRANSFER_SPEED_RATE_UNIT_SHIFT**)
Read the value of FREQUENCY UNIT in TRANSFER SPEED field.
- #define **SD_RD_TRANSFER_SPEED_TIME_VALUE(x)** (((x.transferSpeed) & **SD_TRANSFER_SPEED_TIME_VALUE_MASK**) >> **SD_TRANSFER_SPEED_TIME_VALUE_SHIFT**)

- *Read the value of TIME VALUE in TRANSFER SPEED field.*
• #define **MMC_PRODUCT_NAME_BYTES** (6U)
MMC card product name length united as bytes.
- #define **MMC_SWITCH_COMMAND_SET_SHIFT** (0U)
The bit shift for COMMAND SET field in SWITCH command.
- #define **MMC_SWITCH_COMMAND_SET_MASK** (0x00000007U)
The bit mask for COMMAND set field in SWITCH command.
- #define **MMC_SWITCH_VALUE_SHIFT** (8U)
The bit shift for VALUE field in SWITCH command.
- #define **MMC_SWITCH_VALUE_MASK** (0x0000FF00U)
The bit mask for VALUE field in SWITCH command.
- #define **MMC_SWITCH_BYTE_INDEX_SHIFT** (16U)
The bit shift for BYTE INDEX field in SWITCH command.
- #define **MMC_SWITCH_BYTE_INDEX_MASK** (0x00FF0000U)
The bit mask for BYTE INDEX field in SWITCH command.
- #define **MMC_SWITCH_ACCESS_MODE_SHIFT** (24U)
The bit shift for ACCESS MODE field in SWITCH command.
- #define **MMC_SWITCH_ACCESS_MODE_MASK** (0x03000000U)
The bit mask for ACCESS MODE field in SWITCH command.

Typedefs

- typedef void(* **sd_cd_t**)(bool isInserted, void *userData)
card detect application callback definition
- typedef bool(* **sd_cd_status_t**)(void)
card detect status
- typedef void(* **sd_io_voltage_func_t**)(sdmmc_operation_voltage_t voltage)
card switch voltage function pointer
- typedef void(* **sd_pwr_t**)(bool enable)
card power control function pointer
- typedef void(* **sd_io_strength_t**)(uint32_t busFreq)
card io strength control
- typedef void(* **sdio_int_t**)(void *userData)
card interrupt function pointer

Enumerations

- enum {

kStatus_SDMMC_NotSupportYet = MAKE_STATUS(kStatusGroup_SDMMC, 0U),

kStatus_SDMMC_TransferFailed = MAKE_STATUS(kStatusGroup_SDMMC, 1U),

kStatus_SDMMC_SetCardBlockSizeFailed = MAKE_STATUS(kStatusGroup_SDMMC, 2U),

kStatus_SDMMC_HostNotSupport = MAKE_STATUS(kStatusGroup_SDMMC, 3U),

kStatus_SDMMC_CardNotSupport = MAKE_STATUS(kStatusGroup_SDMMC, 4U),

kStatus_SDMMC_AllSendCidFailed = MAKE_STATUS(kStatusGroup_SDMMC, 5U),

kStatus_SDMMC_SendRelativeAddressFailed = MAKE_STATUS(kStatusGroup_SDMMC, 6U),

kStatus_SDMMC_SendCsdFailed = MAKE_STATUS(kStatusGroup_SDMMC, 7U),

kStatus_SDMMC_SelectCardFailed = MAKE_STATUS(kStatusGroup_SDMMC, 8U),

kStatus_SDMMC_SendScrFailed = MAKE_STATUS(kStatusGroup_SDMMC, 9U),

kStatus_SDMMC_SetDataBusWidthFailed = MAKE_STATUS(kStatusGroup_SDMMC, 10U),

kStatus_SDMMC_GoIdleFailed = MAKE_STATUS(kStatusGroup_SDMMC, 11U),

kStatus_SDMMC_HandShakeOperationConditionFailed,

kStatus_SDMMC_SendApplicationCommandFailed,

kStatus_SDMMC_SwitchFailed = MAKE_STATUS(kStatusGroup_SDMMC, 14U),

kStatus_SDMMC_StopTransmissionFailed = MAKE_STATUS(kStatusGroup_SDMMC, 15U),

kStatus_SDMMC_WaitWriteCompleteFailed = MAKE_STATUS(kStatusGroup_SDMMC, 16U),

kStatus_SDMMC_SetBlockCountFailed = MAKE_STATUS(kStatusGroup_SDMMC, 17U),

kStatus_SDMMC_SetRelativeAddressFailed = MAKE_STATUS(kStatusGroup_SDMMC, 18U),

kStatus_SDMMC_SwitchBusTimingFailed = MAKE_STATUS(kStatusGroup_SDMMC, 19U),

kStatus_SDMMC_SendExtendedCsdFailed = MAKE_STATUS(kStatusGroup_SDMMC, 20U),

kStatus_SDMMC_ConfigureBootFailed = MAKE_STATUS(kStatusGroup_SDMMC, 21U),

kStatus_SDMMC_ConfigureExtendedCsdFailed = MAKE_STATUS(kStatusGroup_SDMMC, 22-U),

kStatus_SDMMC_EnableHighCapacityEraseFailed,

kStatus_SDMMC_SendTestPatternFailed = MAKE_STATUS(kStatusGroup_SDMMC, 24U),

kStatus_SDMMC_ReceiveTestPatternFailed = MAKE_STATUS(kStatusGroup_SDMMC, 25U),

kStatus_SDMMC_SDIO_ResponseError = MAKE_STATUS(kStatusGroup_SDMMC, 26U),

kStatus_SDMMC_SDIO_InvalidArgument,

kStatus_SDMMC_SDIO_SendOperationConditionFail,

kStatus_SDMMC_InvalidVoltage = MAKE_STATUS(kStatusGroup_SDMMC, 29U),

kStatus_SDMMC_SDIO_SwitchHighSpeedFail = MAKE_STATUS(kStatusGroup_SDMMC, 30-U),

kStatus_SDMMC_SDIO_ReadCISFail = MAKE_STATUS(kStatusGroup_SDMMC, 31U),

kStatus_SDMMC_SDIO_InvalidCard = MAKE_STATUS(kStatusGroup_SDMMC, 32U),

kStatus_SDMMC_TuningFail = MAKE_STATUS(kStatusGroup_SDMMC, 33U),

kStatus_SDMMC_SwitchVoltageFail = MAKE_STATUS(kStatusGroup_SDMMC, 34U),

kStatus_SDMMC_SwitchVoltage18VFail33VSuccess = MAKE_STATUS(kStatusGroup_SDMM-

C, 35U),

```
kStatus_SDMMC_ReTuningRequest = MAKE_STATUS(kStatusGroup_SDMMC, 36U),
kStatus_SDMMC_SetDriverStrengthFail = MAKE_STATUS(kStatusGroup_SDMMC, 37U),
kStatus_SDMMC_SetPowerClassFail = MAKE_STATUS(kStatusGroup_SDMMC, 38U),
kStatus_SDMMC_HostNotReady = MAKE_STATUS(kStatusGroup_SDMMC, 39U),
kStatus_SDMMC_CardDetectFailed = MAKE_STATUS(kStatusGroup_SDMMC, 40U),
kStatus_SDMMC_AuSizeNotSetProperly = MAKE_STATUS(kStatusGroup_SDMMC, 41U),
kStatus_SDMMC_PollingCardIdleFailed = MAKE_STATUS(kStatusGroup_SDMMC, 42U),
kStatus_SDMMC_DeselectCardFailed = MAKE_STATUS(kStatusGroup_SDMMC, 43U),
kStatus_SDMMC_CardStatusIdle = MAKE_STATUS(kStatusGroup_SDMMC, 44U),
kStatus_SDMMC_CardStatusBusy = MAKE_STATUS(kStatusGroup_SDMMC, 45U),
kStatus_SDMMC_CardInitFailed = MAKE_STATUS(kStatusGroup_SDMMC, 46U) }
```

SD/MMC card API's running status.

- enum {

kSDMMC_SignalLineCmd = 1U,

kSDMMC_SignalLineData0 = 2U,

kSDMMC_SignalLineData1 = 4U,

kSDMMC_SignalLineData2 = 8U,

kSDMMC_SignalLineData3 = 16U,

kSDMMC_SignalLineData4 = 32U,

kSDMMC_SignalLineData5 = 64U,

kSDMMC_SignalLineData6 = 128U,

kSDMMC_SignalLineData7 = 256U }

sdmmc signal line

- enum **sdmmc_operation_voltage_t** {

kSDMMC_OperationVoltageNone = 0U,

kSDMMC_OperationVoltage330V = 1U,

kSDMMC_OperationVoltage300V = 2U,

kSDMMC_OperationVoltage180V = 3U }

card operation voltage

- enum {

kSDMMC_BusWidth1Bit = 0U,

kSDMMC_BusWidth4Bit = 1U,

kSDMMC_BusWidth8Bit = 2U }

card bus width

- enum { **kSDMMC_Support8BitWidth** = 1U }

sdmmc capability flag

- enum {

kSDMMC_DataPacketFormatLSBFirst,

kSDMMC_DataPacketFormatMSBFirst }

@ brief sdmmc data packet format

- enum **sd_detect_card_type_t** {

kSD_DetectCardByGpioCD,

kSD_DetectCardByHostCD,

kSD_DetectCardByHostDATA3 }

sd card detect type

- enum {

 kSD_Inserted = 1U,

 kSD_Removed = 0U }

 @ brief SD card detect status
- enum {

 kSD_DAT3PullDown = 0U,

 kSD_DAT3PullUp = 1U }

 @ brief SD card detect status
- enum **sd_io_voltage_ctrl_type_t** {

 kSD_IOVoltageCtrlNotSupport = 0U,

 kSD_IOVoltageCtrlByHost = 1U,

 kSD_IOVoltageCtrlByGpio = 2U }

 io voltage control type
- enum {

 kSDMMC_R1OutOfRangeFlag = 31,

 kSDMMC_R1AddressErrorFlag = 30,

 kSDMMC_R1BlockLengthErrorFlag = 29,

 kSDMMC_R1EraseSequenceErrorFlag = 28,

 kSDMMC_R1EraseParameterErrorFlag = 27,

 kSDMMC_R1WriteProtectViolationFlag = 26,

 kSDMMC_R1CardIsLockedFlag = 25,

 kSDMMC_R1LockUnlockFailedFlag = 24,

 kSDMMC_R1CommandCrcErrorFlag = 23,

 kSDMMC_R1IllegalCommandFlag = 22,

 kSDMMC_R1CardEccFailedFlag = 21,

 kSDMMC_R1CardControllerErrorFlag = 20,

 kSDMMC_R1ErrorFlag = 19,

 kSDMMC_R1CidCsdOverwriteFlag = 16,

 kSDMMC_R1WriteProtectEraseSkipFlag = 15,

 kSDMMC_R1CardEccDisabledFlag = 14,

 kSDMMC_R1EraseResetFlag = 13,

 kSDMMC_R1ReadyForDataFlag = 8,

 kSDMMC_R1SwitchErrorFlag = 7,

 kSDMMC_R1ApplicationCommandFlag = 5,

 kSDMMC_R1AuthenticationSequenceErrorFlag = 3 }
- Card status bit in R1.
- enum **sdmmc_r1_current_state_t** {

 kSDMMC_R1StateIdle = 0U,

 kSDMMC_R1StateReady = 1U,

 kSDMMC_R1StateIdentify = 2U,

 kSDMMC_R1StateStandby = 3U,

 kSDMMC_R1StateTransfer = 4U,

 kSDMMC_R1StateSendData = 5U,

 kSDMMC_R1StateReceiveData = 6U,

 kSDMMC_R1StateProgram = 7U,

 kSDMMC_R1StateDisconnect = 8U }

- enum {

 kSDSPI_R1InIdleStateFlag = (1U << 0U),

 kSDSPI_R1EraseResetFlag = (1U << 1U),

 kSDSPI_R1IllegalCommandFlag = (1U << 2U),

 kSDSPI_R1CommandCrcErrorFlag = (1U << 3U),

 kSDSPI_R1EraseSequenceErrorFlag = (1U << 4U),

 kSDSPI_R1AddressErrorFlag = (1U << 5U),

 kSDSPI_R1ParameterErrorFlag = (1U << 6U) }

 Error bit in SPI mode R1.
- enum {

 kSDSPI_R2CardLockedFlag = (1U << 0U),

 kSDSPI_R2WriteProtectEraseSkip = (1U << 1U),

 kSDSPI_R2LockUnlockFailed = (1U << 1U),

 kSDSPI_R2ErrorFlag = (1U << 2U),

 kSDSPI_R2CardControllerErrorFlag = (1U << 3U),

 kSDSPI_R2CardEccFailedFlag = (1U << 4U),

 kSDSPI_R2WriteProtectViolationFlag = (1U << 5U),

 kSDSPI_R2EraseParameterErrorFlag = (1U << 6U),

 kSDSPI_R2OutOfRangeFlag = (1U << 7U),

 kSDSPI_R2CsdOverwriteFlag = (1U << 7U) }

 Error bit in SPI mode R2.
- enum {

 kSDSPI_DataErrorTokenError = (1U << 0U),

 kSDSPI_DataErrorTokenCardControllerError = (1U << 1U),

 kSDSPI_DataErrorTokenCardEccFailed = (1U << 2U),

 kSDSPI_DataErrorTokenOutOfRange = (1U << 3U) }

 Data Error Token mask bit.
- enum **sdspi_data_token_t** {

 kSDSPI_DataTokenBlockRead = 0xFEU,

 kSDSPI_DataTokenSingleBlockWrite = 0xFEU,

 kSDSPI_DataTokenMultipleBlockWrite = 0xFCU,

 kSDSPI_DataTokenStopTransfer = 0xFDU }

 Data Token.
- enum **sdspi_data_response_token_t** {

 kSDSPI_DataResponseTokenAccepted = 0x05U,

 kSDSPI_DataResponseTokenCrcError = 0x0BU,

 kSDSPI_DataResponseTokenWriteError = 0x0DU }

 Data Response Token.
- enum **sd_command_t** {

```

kSD_SendRelativeAddress = 3U,
kSD_Switch = 6U,
kSD_SendInterfaceCondition = 8U,
kSD_VoltageSwitch = 11U,
kSD_SpeedClassControl = 20U,
kSD_EraseWriteBlockStart = 32U,
kSD_EraseWriteBlockEnd = 33U,
kSD_SendTuningBlock = 19U }

```

SD card individual commands.

- enum `sdspi_command_t` { `kSDSPI_CommandCrc` = 59U }

SDSPI individual commands.

- enum `sd_application_command_t` {
`kSD_ApplicationSetBusWidth` = 6U,
`kSD_ApplicationStatus` = 13U,
`kSD_ApplicationSendNumberWriteBlocks` = 22U,
`kSD_ApplicationSetWriteBlockEraseCount` = 23U,
`kSD_ApplicationSendOperationCondition` = 41U,
`kSD_ApplicationSetClearCardDetect` = 42U,
`kSD_ApplicationSendScr` = 51U }

SD card individual application commands.

- enum {
`kSDMMC_CommandClassBasic` = (1U << 0U),
`kSDMMC_CommandClassBlockRead` = (1U << 2U),
`kSDMMC_CommandClassBlockWrite` = (1U << 4U),
`kSDMMC_CommandClassErase` = (1U << 5U),
`kSDMMC_CommandClassWriteProtect` = (1U << 6U),
`kSDMMC_CommandClassLockCard` = (1U << 7U),
`kSDMMC_CommandClassApplicationSpecific` = (1U << 8U),
`kSDMMC_CommandClassInputOutputMode` = (1U << 9U),
`kSDMMC_CommandClassSwitch` = (1U << 10U) }

SD card command class.

- enum {
`kSD_OcrPowerUpBusyFlag` = 31,
`kSD_OcrHostCapacitySupportFlag` = 30,
`kSD_OcrCardCapacitySupportFlag` = `kSD_OcrHostCapacitySupportFlag`,
`kSD_OcrSwitch18RequestFlag` = 24,
`kSD_OcrSwitch18AcceptFlag` = `kSD_OcrSwitch18RequestFlag`,
`kSD_OcrVdd27_28Flag` = 15,
`kSD_OcrVdd28_29Flag` = 16,
`kSD_OcrVdd29_30Flag` = 17,
`kSD_OcrVdd30_31Flag` = 18,
`kSD_OcrVdd31_32Flag` = 19,
`kSD_OcrVdd32_33Flag` = 20,
`kSD_OcrVdd33_34Flag` = 21,
`kSD_OcrVdd34_35Flag` = 22,
`kSD_OcrVdd35_36Flag` = 23 } }

- *OCR register in SD card.*
- enum {

 kSD_SpecificationVersion1_0 = (1U << 0U),

 kSD_SpecificationVersion1_1 = (1U << 1U),

 kSD_SpecificationVersion2_0 = (1U << 2U),

 kSD_SpecificationVersion3_0 = (1U << 3U) }

 SD card specification version number.
- enum **sd_switch_mode_t** {

 kSD_SwitchCheck = 0U,

 kSD_SwitchSet = 1U }

 SD card switch mode.
- enum {

 kSD_CsdReadBlockPartialFlag = (1U << 0U),

 kSD_CsdWriteBlockMisalignFlag = (1U << 1U),

 kSD_CsdReadBlockMisalignFlag = (1U << 2U),

 kSD_CsdDsrImplementedFlag = (1U << 3U),

 kSD_CsdEraseBlockEnabledFlag = (1U << 4U),

 kSD_CsdWriteProtectGroupEnabledFlag = (1U << 5U),

 kSD_CsdWriteBlockPartialFlag = (1U << 6U),

 kSD_CsdFileFormatGroupFlag = (1U << 7U),

 kSD_CsdCopyFlag = (1U << 8U),

 kSD_CsdPermanentWriteProtectFlag = (1U << 9U),

 kSD_CsdTemporaryWriteProtectFlag = (1U << 10U) }

 SD card CSD register flags.
- enum {

 kSD_ScrDataStatusAfterErase = (1U << 0U),

 kSD_ScrSdSpecification3 = (1U << 1U) }

 SD card SCR register flags.
- enum {

 kSD_FunctionSDR12Default = 0U,

 kSD_FunctionSDR25HighSpeed = 1U,

 kSD_FunctionSDR50 = 2U,

 kSD_FunctionSDR104 = 3U,

 kSD_FunctionDDR50 = 4U }

 SD timing function number.
- enum {

 kSD_GroupTimingMode = 0U,

 kSD_GroupCommandSystem = 1U,

 kSD_GroupDriverStrength = 2U,

 kSD_GroupCurrentLimit = 3U }

 SD group number.
- enum **sd_timing_mode_t** {

 kSD_TimingSDR12DefaultMode = 0U,

 kSD_TimingSDR25HighSpeedMode = 1U,

 kSD_TimingSDR50Mode = 2U,

 kSD_TimingSDR104Mode = 3U,

```

kSD_TimingDDR50Mode = 4U }

    SD card timing mode flags.

• enum sd_driver_strength_t {
    kSD_DriverStrengthTypeB = 0U,
    kSD_DriverStrengthTypeA = 1U,
    kSD_DriverStrengthTypeC = 2U,
    kSD_DriverStrengthTypeD = 3U }

    SD card driver strength.

• enum sd_max_current_t {
    kSD_CurrentLimit200MA = 0U,
    kSD_CurrentLimit400MA = 1U,
    kSD_CurrentLimit600MA = 2U,
    kSD_CurrentLimit800MA = 3U }

    SD card current limit.

• enum sdmmc_command_t {
    kSDMMC_GoIdleState = 0U,
    kSDMMC_AllSendCid = 2U,
    kSDMMC_SetDsr = 4U,
    kSDMMC_SelectCard = 7U,
    kSDMMC_SendCsd = 9U,
    kSDMMC_SendCid = 10U,
    kSDMMC_StopTransmission = 12U,
    kSDMMC_SendStatus = 13U,
    kSDMMC_GoInactiveState = 15U,
    kSDMMC_SetBlockLength = 16U,
    kSDMMC_ReadSingleBlock = 17U,
    kSDMMC_ReadMultipleBlock = 18U,
    kSDMMC_SetBlockCount = 23U,
    kSDMMC_WriteSingleBlock = 24U,
    kSDMMC_WriteMultipleBlock = 25U,
    kSDMMC_ProgramCsd = 27U,
    kSDMMC_SetWriteProtect = 28U,
    kSDMMC_ClearWriteProtect = 29U,
    kSDMMC_SendWriteProtect = 30U,
    kSDMMC_Erase = 38U,
    kSDMMC_LockUnlock = 42U,
    kSDMMC_ApplicationCommand = 55U,
    kSDMMC_GeneralCommand = 56U,
    kSDMMC_ReadOcr = 58U }

    SD/MMC card common commands.

• enum {

```

```

kSDIO_RegCCCRSdioVer = 0x00U,
kSDIO_RegSDVersion = 0x01U,
kSDIO_RegIOEnable = 0x02U,
kSDIO_RegIOReady = 0x03U,
kSDIO_RegIOIntEnable = 0x04U,
kSDIO_RegIOIntPending = 0x05U,
kSDIO_RegIOAbort = 0x06U,
kSDIO_RegBusInterface = 0x07U,
kSDIO_RegCardCapability = 0x08U,
kSDIO_RegCommonCISPointer = 0x09U,
kSDIO_RegBusSuspend = 0x0C,
kSDIO_RegFunctionSelect = 0x0DU,
kSDIO_RegExecutionFlag = 0x0EU,
kSDIO_RegReadyFlag = 0x0FU,
kSDIO_RegFN0BlockSizeLow = 0x10U,
kSDIO_RegFN0BlockSizeHigh = 0x11U,
kSDIO_RegPowerControl = 0x12U,
kSDIO_RegBusSpeed = 0x13U,
kSDIO_RegUHSITimingSupport = 0x14U,
kSDIO_RegDriverStrength = 0x15U,
kSDIO_RegInterruptExtension = 0x16U }

    sdio card cccr register addr
• enum sdio_command_t {
    kSDIO_SendRelativeAddress = 3U,
    kSDIO_SendOperationCondition = 5U,
    kSDIO_SendInterfaceCondition = 8U,
    kSDIO_RWIODirect = 52U,
    kSDIO_RWIOExtended = 53U }

    sdio card individual commands
• enum sdio_func_num_t {
    kSDIO_FunctionNum0,
    kSDIO_FunctionNum1,
    kSDIO_FunctionNum2,
    kSDIO_FunctionNum3,
    kSDIO_FunctionNum4,
    kSDIO_FunctionNum5,
    kSDIO_FunctionNum6,
    kSDIO_FunctionNum7,
    kSDIO_FunctionMemory }

    sdio card individual commands
• enum {

```

- ```

kSDIO_StatusCmdCRCError = 0x8000U,
kSDIO_StatusIllegalCmd = 0x4000U,
kSDIO_StatusR6Error = 0x2000U,
kSDIO_StatusError = 0x0800U,
kSDIO_StatusFunctionNumError = 0x0200U,
kSDIO_StatusOutOfRange = 0x0100U }

```

*sdio command response flag*
- enum {
 

```

kSDIO_OcrPowerUpBusyFlag = 31,
kSDIO_OcrIONumber = 28,
kSDIO_OcrMemPresent = 27,
kSDIO_OcrVdd20_21Flag = 8,
kSDIO_OcrVdd21_22Flag = 9,
kSDIO_OcrVdd22_23Flag = 10,
kSDIO_OcrVdd23_24Flag = 11,
kSDIO_OcrVdd24_25Flag = 12,
kSDIO_OcrVdd25_26Flag = 13,
kSDIO_OcrVdd26_27Flag = 14,
kSDIO_OcrVdd27_28Flag = 15,
kSDIO_OcrVdd28_29Flag = 16,
kSDIO_OcrVdd29_30Flag = 17,
kSDIO_OcrVdd30_31Flag = 18,
kSDIO_OcrVdd31_32Flag = 19,
kSDIO_OcrVdd32_33Flag = 20,
kSDIO_OcrVdd33_34Flag = 21,
kSDIO_OcrVdd34_35Flag = 22,
kSDIO_OcrVdd35_36Flag = 23 }

```

*sdio operation condition flag*
- enum {
 

```

kSDIO_CCCRSupportDirectCmdDuringDataTrans = (1UL << 0U),
kSDIO_CCCRSupportMultiBlock = (1UL << 1U),
kSDIO_CCCRSupportReadWait = (1UL << 2U),
kSDIO_CCCRSupportSuspendResume = (1UL << 3U),
kSDIO_CCCRSupportIntDuring4BitDataTrans = (1UL << 4U),
kSDIO_CCCRSupportLowSpeed1Bit = (1UL << 6U),
kSDIO_CCCRSupportLowSpeed4Bit = (1UL << 7U),
kSDIO_CCCRSupportMasterPowerControl = (1UL << 8U),
kSDIO_CCCRSupportHighSpeed = (1UL << 9U),
kSDIO_CCCRSupportContinuousSPIInt = (1UL << 10U) }

```

*sdio capability flag*
- enum {
 

```

kSDIO_FBRSupportCSA = (1U << 0U),
kSDIO_FBRSupportPowerSelection = (1U << 1U) }

```

*sdio fbr flag*
- enum **sdio\_bus\_width\_t** {

- ```

kSDIO_DataBus1Bit = 0x00U,
kSDIO_DataBus4Bit = 0X02U,
kSDIO_DataBus8Bit = 0X03U }
    
```

sdio bus width
- enum `mmc_command_t` {


```

kMMC_SendOperationCondition = 1U,
kMMC_SetRelativeAddress = 3U,
kMMC_SleepAwake = 5U,
kMMC_Switch = 6U,
kMMC_SendExtendedCsd = 8U,
kMMC_ReadDataUntilStop = 11U,
kMMC_BusTestRead = 14U,
kMMC_SendingBusTest = 19U,
kMMC_WriteDataUntilStop = 20U,
kMMC_SendTuningBlock = 21U,
kMMC_ProgramCid = 26U,
kMMC_EraseGroupStart = 35U,
kMMC_EraseGroupEnd = 36U,
kMMC_FastInputOutput = 39U,
kMMC_GoInterruptState = 40U }
    
```

MMC card individual commands.
- enum `mmc_classified_voltage_t` {


```

kMMC_ClassifiedVoltageHigh = 0U,
kMMC_ClassifiedVoltageDual = 1U }
    
```

MMC card classified as voltage range.
- enum `mmc_classified_density_t` { `kMMC_ClassifiedDensityWithin2GB` = 0U }
- enum `mmc_access_mode_t` {


```

kMMC_AccessModeByte = 0U,
kMMC_AccessModeSector = 2U }
    
```

MMC card access mode(Access mode in OCR).
- enum `mmc_voltage_window_t` {


```

kMMC_VoltageWindowNone = 0U,
kMMC_VoltageWindow120 = 0x01U,
kMMC_VoltageWindow170to195 = 0x02U,
kMMC_VoltageWindows270to360 = 0x1FFU }
    
```

MMC card voltage window(VDD voltage window in OCR).
- enum `mmc_csd_structure_version_t` {


```

kMMC_CsdStrucureVersion10 = 0U,
kMMC_CsdStrucureVersion11 = 1U,
kMMC_CsdStrucureVersion12 = 2U,
kMMC_CsdStrucureVersionInExtcsd = 3U }
    
```

CSD structure version(CSD_STRUCTURE in CSD).
- enum `mmc_specification_version_t` { }

```
kMMC_SpecificationVersion0 = 0U,
kMMC_SpecificationVersion1 = 1U,
kMMC_SpecificationVersion2 = 2U,
kMMC_SpecificationVersion3 = 3U,
kMMC_SpecificationVersion4 = 4U }
```

MMC card specification version(SPEC_VERS in CSD).

- enum {

kMMC_ExtendedCsdRevision10 = 0U,
 kMMC_ExtendedCsdRevision11 = 1U,
 kMMC_ExtendedCsdRevision12 = 2U,
 kMMC_ExtendedCsdRevision13 = 3U,
 kMMC_ExtendedCsdRevision14 = 4U,
 kMMC_ExtendedCsdRevision15 = 5U,
 kMMC_ExtendedCsdRevision16 = 6U,
 kMMC_ExtendedCsdRevision17 = 7U }

MMC card Extended CSD fix version(EXT_CSD_REV in Extended CSD)

- enum **mmc_command_set_t** {

kMMC_CommandSetStandard = 0U,
 kMMC_CommandSet1 = 1U,
 kMMC_CommandSet2 = 2U,
 kMMC_CommandSet3 = 3U,
 kMMC_CommandSet4 = 4U }

MMC card command set(COMMAND_SET in Extended CSD)

- enum {

kMMC_SupportAlternateBoot = 1U,
 kMMC_SupportDDRBoot = 2U,
 kMMC_SupportHighSpeedBoot = 4U }

- boot support(BOOT_INFO in Extended CSD)
- enum **mmc_high_speed_timing_t** {

kMMC_HighSpeedTimingNone = 0U,
 kMMC_HighSpeedTiming = 1U,
 kMMC_HighSpeed200Timing = 2U,
 kMMC_HighSpeed400Timing = 3U,
 kMMC_EnhanceHighSpeed400Timing = 4U }

MMC card high-speed timing(HS_TIMING in Extended CSD)

- enum **mmc_data_bus_width_t** {

kMMC_DataBusWidth1bit = 0U,
 kMMC_DataBusWidth4bit = 1U,
 kMMC_DataBusWidth8bit = 2U,
 kMMC_DataBusWidth4bitDDR = 5U,
 kMMC_DataBusWidth8bitDDR = 6U,
 kMMC_DataBusWidth8bitDDRSTROBE = 0x86U }

MMC card data bus width(BUS_WIDTH in Extended CSD)

- enum **mmc_boot_partition_enable_t** {

```
kMMC_BootPartitionEnableNot = 0U,
kMMC_BootPartitionEnablePartition1 = 1U,
kMMC_BootPartitionEnablePartition2 = 2U,
kMMC_BootPartitionEnableUserAera = 7U }
```

MMC card boot partition enabled(BOOT_PARTITION_ENABLE in Extended CSD)

- enum `mmc_boot_timing_mode_t` {


```
kMMC_BootModeSDRWithDefaultTiming = 0U,
kMMC_BootModeSDRWithHighSpeedTiming = 1U,
kMMC_BootModeDDRTiming = 2U }
```

boot mode configuration Note: HS200 & HS400 is not support during BOOT operation.
- enum `mmc_boot_partition_wp_t` {


```
kMMC_BootPartitionWPDisable = 0x50U,
kMMC_BootPartitionPwrWPToBothPartition,
kMMC_BootPartitionPermWPToBothPartition = 0x04U,
kMMC_BootPartitionPwrWPToPartition1 = (1U << 7U) | 1U,
kMMC_BootPartitionPwrWPToPartition2 = (1U << 7U) | 3U,
kMMC_BootPartitionPermWPToPartition1,
kMMC_BootPartitionPermWPToPartition2,
kMMC_BootPartitionPermWPToPartition1PwrWPToPartition2,
kMMC_BootPartitionPermWPToPartition2PwrWPToPartition1 }
```

MMC card boot partition write protect configurations All the bits in BOOT_WP register, except the two R/W bits B_PERM_WP_DIS and B_PERM_WP_EN, shall only be written once per power cycle. The protection mode intended for both boot areas will be set with a single write.

- enum {


```
kMMC_BootPartitionNotProtected = 0U,
kMMC_BootPartitionPwrProtected = 1U,
kMMC_BootPartitionPermProtected = 2U }
```

MMC card boot partition write protect status.

- enum `mmc_access_partition_t` {


```
kMMC_AccessPartitionUserAera = 0U,
kMMC_AccessPartitionBoot1 = 1U,
kMMC_AccessPartitionBoot2 = 2U,
kMMC_AccessRPMB = 3U,
kMMC_AccessGeneralPurposePartition1 = 4U,
kMMC_AccessGeneralPurposePartition2 = 5U,
kMMC_AccessGeneralPurposePartition3 = 6U,
kMMC_AccessGeneralPurposePartition4 = 7U }
```

MMC card partition to be accessed(BOOT_PARTITION_ACCESS in Extended CSD)
- enum {

```

kMMC_CsdReadBlockPartialFlag = (1U << 0U),
kMMC_CsdWriteBlockMisalignFlag = (1U << 1U),
kMMC_CsdReadBlockMisalignFlag = (1U << 2U),
kMMC_CsdDsrImplementedFlag = (1U << 3U),
kMMC_CsdWriteProtectGroupEnabledFlag = (1U << 4U),
kMMC_CsdWriteBlockPartialFlag = (1U << 5U),
kMMC_ContentProtectApplicationFlag = (1U << 6U),
kMMC_CsdFileFormatGroupFlag = (1U << 7U),
kMMC_CsdCopyFlag = (1U << 8U),
kMMC_CsdPermanentWriteProtectFlag = (1U << 9U),
kMMC_CsdTemporaryWriteProtectFlag = (1U << 10U) }

```

MMC card CSD register flags.

- enum `mmc_extended_csd_access_mode_t` {

kMMC_ExtendedCsdAccessModeCommandSet = 0U,

kMMC_ExtendedCsdAccessModeSetBits = 1U,

kMMC_ExtendedCsdAccessModeClearBits = 2U,

kMMC_ExtendedCsdAccessModeWriteBits = 3U }

Extended CSD register access mode(Access mode in CMD6).

- enum `mmc_extended_csd_index_t` {

kMMC_ExtendedCsdIndexFlushCache = 32U,

kMMC_ExtendedCsdIndexCacheControl = 33U,

kMMC_ExtendedCsdIndexBootPartitionWP = 173U,

kMMC_ExtendedCsdIndexEraseGroupDefinition = 175U,

kMMC_ExtendedCsdIndexBootBusConditions = 177U,

kMMC_ExtendedCsdIndexBootConfigWP = 178U,

kMMC_ExtendedCsdIndexPartitionConfig = 179U,

kMMC_ExtendedCsdIndexBusWidth = 183U,

kMMC_ExtendedCsdIndexHighSpeedTiming = 185U,

kMMC_ExtendedCsdIndexPowerClass = 187U,

kMMC_ExtendedCsdIndexCommandSet = 191U }

EXT CSD byte index.

- enum {

kMMC_DriverStrength0 = 0U,

kMMC_DriverStrength1 = 1U,

kMMC_DriverStrength2 = 2U,

kMMC_DriverStrength3 = 3U,

kMMC_DriverStrength4 = 4U }
- mmc driver strength*
- enum `mmc_extended_csd_flags_t` {

kMMC_ExtCsdExtPartitionSupport = (1 << 0U),

kMMC_ExtCsdEnhancePartitionSupport = (1 << 1U),

kMMC_ExtCsdPartitioningSupport = (1 << 2U),

kMMC_ExtCsdPrgCIDCSDInDDRModeSupport = (1 << 3U),

kMMC_ExtCsdBKOpsSupport = (1 << 4U),

kMMC_ExtCsdDataTagSupport = (1 << 5U),

kMMC_ExtCsdModeOperationCodeSupport = (1 << 6U) }

- enum `mmc_boot_mode_t` {

 `kMMC_BootModeNormal` = 0U,

 `kMMC_BootModeAlternative` = 1U }

 MMC card boot mode.

common function

tuning pattern

- `status_t SDMMC_SelectCard` (`sdmmchost_t *host`, `uint32_t relativeAddress`, `bool isSelected`)

 Selects the card to put it into transfer state.
- `status_t SDMMC_SendApplicationCommand` (`sdmmchost_t *host`, `uint32_t relativeAddress`)

 Sends an application command.
- `status_t SDMMC_SetBlockCount` (`sdmmchost_t *host`, `uint32_t blockCount`)

 Sets the block count.
- `status_t SDMMC_GoIdle` (`sdmmchost_t *host`)

 Sets the card to be idle state.
- `status_t SDMMC_SetBlockSize` (`sdmmchost_t *host`, `uint32_t blockSize`)

 Sets data block size.
- `status_t SDMMC_SetCardInactive` (`sdmmchost_t *host`)

 Sets card to inactive status.

38.3.2 Data Structure Documentation

38.3.2.1 struct sd_detect_card_t

Data Fields

- `sd_detect_card_type_t type`

 card detect type
- `uint32_t cdDebounce_ms`

 card detect debounce delay ms
- `sd_cd_t callback`

 card inserted callback which is meaningful for interrupt case
- `sd_cd_status_t cardDetected`

 used to check sd cd status when card detect through GPIO
- `sd_dat3_pull_t dat3PullFunc`

 function pointer of DATA3 pull up/down
- `void * userData`

 user data

38.3.2.2 struct sd_io_voltage_t

Data Fields

- `sd_io_voltage_ctrl_type_t type`

- **sd_io_voltage_func_t func**
io voltage switch function

38.3.2.3 struct sd_usr_param_t

Data Fields

- **sd_pwr_t pwr**
power control configuration pointer
- **uint32_t powerOnDelayMS**
power on delay time
- **uint32_t powerOffDelayMS**
power off delay time
- **sd_io_strength_t ioStrength**
switch sd io strength
- **sd_io_voltage_t * ioVoltage**
switch io voltage
- **sd_detect_card_t * cd**
card detect
- **uint32_t maxFreq**
board support maximum frequency
- **uint32_t capability**
board capability flag

38.3.2.4 struct sdio_card_int_t

Data Fields

- **void * userData**
user data
- **sdio_int_t cardInterrupt**
card int call back

38.3.2.5 struct sdio_usr_param_t

Data Fields

- **sd_pwr_t pwr**
power control configuration pointer
- **uint32_t powerOnDelayMS**
power on delay time
- **uint32_t powerOffDelayMS**
power off delay time
- **sd_io_strength_t ioStrength**
switch sd io strength
- **sd_io_voltage_t * ioVoltage**
switch io voltage

- `sd_detect_card_t * cd`
card detect
- `sdio_card_int_t * sdioInt`
card int
- `uint32_t maxFreq`
board support maximum frequency
- `uint32_t capability`
board capability flag

38.3.2.6 struct `sdio_fbr_t`

Data Fields

- `uint8_t flags`
current io flags
- `uint8_t ioStdFunctionCode`
current io standard function code
- `uint8_t ioExtFunctionCode`
current io extended function code
- `uint32_t ioPointerToCIS`
current io pointer to CIS
- `uint32_t ioPointerToCSA`
current io pointer to CSA
- `uint16_t ioBlockSize`
current io block size

38.3.2.7 struct `sdio_common_cis_t`

Data Fields

- `uint16_t mID`
manufacturer code
- `uint16_t mInfo`
manufacturer information
- `uint8_t funcID`
function ID
- `uint16_t fn0MaxBlkSize`
function 0 max block size
- `uint8_t maxTransSpeed`
max data transfer speed for all function

38.3.2.8 struct `sdio_func_cis_t`

Data Fields

- `uint8_t funcID`
function ID
- `uint8_t funcInfo`

- *function info*
- **uint8_t ioVersion**
level of application specification this io support
- **uint32_t cardPSN**
product serial number
- **uint32_t ioCSASize**
available CSA size for io
- **uint8_t ioCSAProperty**
CSA property.
- **uint16_t ioMaxBlockSize**
io max transfer data size
- **uint32_t ioOCR**
io ioeration condition
- **uint8_t ioOPMinPwr**
min current in operation mode
- **uint8_t ioOPAvgPwr**
average current in operation mode
- **uint8_t ioOPMaxPwr**
max current in operation mode
- **uint8_t ioSBMinPwr**
min current in standby mode
- **uint8_t ioSBAvgPwr**
average current in standby mode
- **uint8_t ioSBMaxPwr**
max current in standby mode
- **uint16_t ioMinBandWidth**
io min transfer bandwidth
- **uint16_t ioOptimumBandWidth**
io optimum transfer bandwidth
- **uint16_t ioReadyTimeout**
timeout value from enalbe to ready
- **uint16_t ioHighCurrentAvgCurrent**
the average peak current (mA)
when IO operating in high current mode
- **uint16_t ioHighCurrentMaxCurrent**
the max peak current (mA)
when IO operating in high current mode
- **uint16_t ioLowCurrentAvgCurrent**
the average peak current (mA)
when IO operating in lower current mode
- **uint16_t ioLowCurrentMaxCurrent**
the max peak current (mA)
when IO operating in lower current mode

38.3.2.9 struct sd_status_t

Data Fields

- **uint8_t busWidth**
current buswidth

- `uint8_t secureMode`
secured mode
- `uint16_t cardType`
sdcard type
- `uint32_t protectedSize`
size of protected area
- `uint8_t speedClass`
speed class of card
- `uint8_t performanceMove`
Performance of move indicated by 1[MB/S]step.
- `uint8_t auSize`
size of AU
- `uint16_t eraseSize`
number of AUs to be erased at a time
- `uint8_t eraseTimeout`
timeout value for erasing areas specified by UNIT OF ERASE AU
- `uint8_t eraseOffset`
fixed offset value added to erase time
- `uint8_t uhsSpeedGrade`
speed grade for UHS mode
- `uint8_t uhsAuSize`
size of AU for UHS mode

38.3.2.10 struct sd_cid_t

Data Fields

- `uint8_t manufacturerID`
Manufacturer ID [127:120].
- `uint16_t applicationID`
OEM/Application ID [119:104].
- `uint8_t productName [SD_PRODUCT_NAME_BYTES]`
Product name [103:64].
- `uint8_t productVersion`
Product revision [63:56].
- `uint32_t productSerialNumber`
Product serial number [55:24].
- `uint16_t manufacturerData`
Manufacturing date [19:8].

38.3.2.11 struct sd_csd_t

Data Fields

- `uint8_t csdStructure`
CSD structure [127:126].
- `uint8_t dataReadAccessTime1`
Data read access-time-1 [119:112].
- `uint8_t dataReadAccessTime2`

- **uint8_t transferSpeed**
Maximum data transfer rate [103:96].
- **uint16_t cardCommandClass**
Card command classes [95:84].
- **uint8_t readBlockLength**
Maximum read data block length [83:80].
- **uint16_t flags**
Flags in _sd_csd_flag.
- **uint32_t deviceSize**
Device size [73:62].
- **uint8_t readCurrentVddMin**
Maximum read current at VDD min [61:59].
- **uint8_t readCurrentVddMax**
Maximum read current at VDD max [58:56].
- **uint8_t writeCurrentVddMin**
Maximum write current at VDD min [55:53].
- **uint8_t writeCurrentVddMax**
Maximum write current at VDD max [52:50].
- **uint8_t deviceSizeMultiplier**
Device size multiplier [49:47].
- **uint8_t eraseSectorSize**
Erase sector size [45:39].
- **uint8_t writeProtectGroupSize**
Write protect group size [38:32].
- **uint8_t writeSpeedFactor**
Write speed factor [28:26].
- **uint8_t writeBlockLength**
Maximum write data block length [25:22].
- **uint8_t fileFormat**
File format [11:10].

38.3.2.12 struct sd_scr_t

Data Fields

- **uint8_t scrStructure**
SCR Structure [63:60].
- **uint8_t sdSpecification**
SD memory card specification version [59:56].
- **uint16_t flags**
SCR flags in _sd_scr_flag.
- **uint8_t sdSecurity**
Security specification supported [54:52].
- **uint8_t sdBusWidths**
Data bus widths supported [51:48].
- **uint8_t extendedSecurity**
Extended security support [46:43].
- **uint8_t commandSupport**
Command support bits [33:32] 33-support CMD23, 32-support cmd20.

- `uint32_t reservedForManufacturer`
reserved for manufacturer usage [31:0]

38.3.2.13 struct mmc_cid_t

Data Fields

- `uint8_t manufacturerID`
Manufacturer ID.
- `uint16_t applicationID`
OEM/Application ID.
- `uint8_t productName [MMC_PRODUCT_NAME_BYTES]`
Product name.
- `uint8_t productVersion`
Product revision.
- `uint32_t productSerialNumber`
Product serial number.
- `uint8_t manufacturerData`
Manufacturing date.

38.3.2.14 struct mmc_csd_t

Data Fields

- `uint8_t csdStructureVersion`
CSD structure [127:126].
- `uint8_t systemSpecificationVersion`
System specification version [125:122].
- `uint8_t dataReadAccessTime1`
Data read access-time 1 [119:112].
- `uint8_t dataReadAccessTime2`
*Data read access-time 2 in CLOCK cycles (NSAC*100) [111:104].*
- `uint8_t transferSpeed`
Max.
- `uint16_t cardCommandClass`
card command classes [95:84]
- `uint8_t readBlockLength`
Max.
- `uint16_t flags`
Contain flags in _mmc_csd_flag.
- `uint16_t deviceSize`
Device size [73:62].
- `uint8_t readCurrentVddMin`
Max.
- `uint8_t readCurrentVddMax`
Max.
- `uint8_t writeCurrentVddMin`
Max.
- `uint8_t writeCurrentVddMax`

- **uint8_t deviceSizeMultiplier**
Device size multiplier [49:47].
- **uint8_t eraseGroupSize**
Erase group size [46:42].
- **uint8_t eraseGroupSizeMultiplier**
Erase group size multiplier [41:37].
- **uint8_t writeProtectGroupSize**
Write protect group size [36:32].
- **uint8_t defaultEcc**
Manufacturer default ECC [30:29].
- **uint8_t writeSpeedFactor**
Write speed factor [28:26].
- **uint8_t maxWriteBlockLength**
Max.
- **uint8_t fileFormat**
File format [11:10].
- **uint8_t eccCode**
ECC code [9:8].

Field Documentation

(1) **uint8_t mmc_csd_t::transferSpeed**

bus clock frequency [103:96]

(2) **uint8_t mmc_csd_t::readBlockLength**

read data block length [83:80]

(3) **uint8_t mmc_csd_t::readCurrentVddMin**

read current @ VDD min [61:59]

(4) **uint8_t mmc_csd_t::readCurrentVddMax**

read current @ VDD max [58:56]

(5) **uint8_t mmc_csd_t::writeCurrentVddMin**

write current @ VDD min [55:53]

(6) **uint8_t mmc_csd_t::writeCurrentVddMax**

write current @ VDD max [52:50]

(7) **uint8_t mmc_csd_t::maxWriteBlockLength**

write data block length [25:22]

38.3.2.15 struct mmc_extended_csd_t

Data Fields

- uint8_t **cacheCtrl**
< secure removal type[16]
- uint8_t **partitionAttribute**
< power off notification[34]
- uint8_t **userWP**
< max enhance area size [159-157]
- uint8_t **bootPartitionWP**
boot write protect register[173]
- uint8_t **bootWPStatus**
boot write protect status register[174]
- uint8_t **highDensityEraseGroupDefinition**
High-density erase group definition [175].
- uint8_t **bootDataBusConditions**
Boot bus conditions [177].
- uint8_t **bootConfigProtect**
Boot config protection [178].
- uint8_t **partitionConfig**
Boot configuration [179].
- uint8_t **eraseMemoryContent**
Erasered memory content [181].
- uint8_t **dataBusWidth**
Data bus width mode [183].
- uint8_t **highSpeedTiming**
High-speed interface timing [185].
- uint8_t **powerClass**
Power class [187].
- uint8_t **commandSetRevision**
Command set revision [189].
- uint8_t **commandSet**
Command set [191].
- uint8_t **extendecCsdVersion**
Extended CSD revision [192].
- uint8_t **csdStructureVersion**
CSD structure version [194].
- uint8_t **cardType**
Card Type [196].
- uint8_t **ioDriverStrength**
IO driver strength [197].
- uint8_t **partitionSwitchTimeout**
< out of interrupt busy timing [198]
- uint8_t **powerClass52MHz195V**
Power Class for 52MHz @ 1.95V [200].
- uint8_t **powerClass26MHz195V**
Power Class for 26MHz @ 1.95V [201].
- uint8_t **powerClass52MHz360V**
Power Class for 52MHz @ 3.6V [202].
- uint8_t **powerClass26MHz360V**

- **Power Class for 26MHz @ 3.6V [203].**
- **uint8_t minimumReadPerformance4Bit26MHz**
Minimum Read Performance for 4bit at 26MHz [205].
- **uint8_t minimumWritePerformance4Bit26MHz**
Minimum Write Performance for 4bit at 26MHz [206].
- **uint8_t minimumReadPerformance8Bit26MHz4Bit52MHz**
Minimum read Performance for 8bit at 26MHz/4bit @ 52MHz [207].
- **uint8_t minimumWritePerformance8Bit26MHz4Bit52MHz**
Minimum Write Performance for 8bit at 26MHz/4bit @ 52MHz [208].
- **uint8_t minimumReadPerformance8Bit52MHz**
Minimum Read Performance for 8bit at 52MHz [209].
- **uint8_t minimumWritePerformance8Bit52MHz**
Minimum Write Performance for 8bit at 52MHz [210].
- **uint32_t sectorCount**
Sector Count [215:212].
- **uint8_t sleepAwakeTimeout**
< sleep notification timeout [216]
- **uint8_t sleepCurrentVCCQ**
< Production state awareness timeout [218]
- **uint8_t sleepCurrentVCC**
Sleep current (VCC) [220].
- **uint8_t highCapacityWriteProtectGroupSize**
High-capacity write protect group size [221].
- **uint8_t reliableWriteSectorCount**
Reliable write sector count [222].
- **uint8_t highCapacityEraseTimeout**
High-capacity erase timeout [223].
- **uint8_t highCapacityEraseUnitSize**
High-capacity erase unit size [224].
- **uint8_t accessSize**
Access size [225].
- **uint8_t minReadPerformance8bitAt52MHZDDR**
< secure trim multiplier[229]
- **uint8_t minWritePerformance8bitAt52MHZDDR**
Minimum write performance for 8bit at DDR 52MHZ[235].
- **uint8_t powerClass200MHZVCCQ130VVCC360V**
power class for 200MHZ, at VCCQ= 1.3V,VCC=3.6V[236]
- **uint8_t powerClass200MHZVCCQ195VVCC360V**
power class for 200MHZ, at VCCQ= 1.95V,VCC=3.6V[237]
- **uint8_t powerClass52MHZDDR195V**
power class for 52MHZ,DDR at Vcc 1.95V[238]
- **uint8_t powerClass52MHZDDR360V**
power class for 52MHZ,DDR at Vcc 3.6V[239]
- **uint32_t genericCMD6Timeout**
< 1st initialization time after partitioning[241]
- **uint32_t cacheSize**
cache size[252-249]
- **uint8_t powerClass200MHZDDR360V**
power class for 200MHZ, DDR at VCC=2.6V[253]
- **uint8_t extPartitionSupport**
< fw VERSION [261-254]

- `uint8_t supportedCommandSet`
< large unit size[495]

Field Documentation

(1) `uint8_t mmc_extended_csd_t::cacheCtrl`

< product state awareness enablement[17]
< max preload data size[21-18]
< pre-load data size[25-22]
< FFU status [26]
< mode operation code[29]
< mode config [30] control to turn on/off cache[33]

(2) `uint8_t mmc_extended_csd_t::partitionAttribute`

< packed cmd fail index [35]
< packed cmd status[36]
< context configuration[51-37]
< extended partitions attribut[53-52]
< exception events status[55-54]
< exception events control[57-56]
< number of group to be released[58]
< class 6 command control[59]
< 1st initialization after disabling sector size emu[60]
< sector size[61]
< sector size emulation[62]
< native sector size[63]
< period wakeup [131]
< package case temperature is controlled[132]
< production state awareness[133]
< enhanced user data start addr [139-136]
< enhanced user data area size[142-140]
< general purpose partition size[154-143] partition attribute [156]

(3) `uint8_t mmc_extended_csd_t::userWP`

< HPI management [161]

< write reliability parameter register[166]
< write reliability setting register[167]
< RPMB size multi [168]
< FW configuration[169] user write protect register[171]

(4) uint8_t mmc_extended_csd_t::partitionSwitchTimeout

partition switch timing [199]

(5) uint8_t mmc_extended_csd_t::sleepAwakeTimeout

Sleep/awake timeout [217]

(6) uint8_t mmc_extended_csd_t::sleepCurrentVCCQ

Sleep current (VCCQ) [219]

(7) uint8_t mmc_extended_csd_t::minReadPerformance8bitAt52MHZDDR

< secure erase multiplier[230]
< secure feature support[231]
< trim multiplier[232] Minimum read performance for 8bit at DDR 52MHZ[234]

(8) uint32_t mmc_extended_csd_t::genericCMD6Timeout

< correct prg sectors number[245-242]
< background operations status[246]
< power off notification timeout[247] generic CMD6 timeout[248]

(9) uint8_t mmc_extended_csd_t::extPartitionSupport

< device version[263-262]
< optimal trim size[264]
< optimal write size[265]
< optimal read size[266]
< pre EOL information[267]
< device life time estimation typeA[268]
< device life time estimation typeB[269]
< number of FW sectors correctly programmed[305-302]
< FFU argument[490-487]
< operation code timeout[491]

< support mode [493] extended partition attribute support[494]

(10) uint8_t mmc_extended_csd_t::supportedCommandSet

< context management capability[496]

< tag resource size[497]

< tag unit size[498]

< max packed write cmd[500]

< max packed read cmd[501]

< HPI feature[503] Supported Command Sets [504]

38.3.2.16 struct mmc_extended_csd_config_t

Data Fields

- **mmc_command_set_t commandSet**
Command set.
- **uint8_t ByteValue**
The value to set.
- **uint8_t ByteIndex**
The byte index in Extended CSD(mmc_extended_csd_index_t)
- **mmc_extended_csd_access_mode_t accessMode**
Access mode.

38.3.2.17 struct mmc_boot_config_t

Data Fields

- **mmc_boot_mode_t bootMode**
mmc boot mode
- **bool enableBootAck**
Enable boot ACK.
- **mmc_boot_partition_enable_t bootPartition**
Boot partition.
- **mmc_boot_timing_mode_t bootTimingMode**
boot mode
- **mmc_data_bus_width_t bootDataBusWidth**
Boot data bus width.
- **bool retainBootbusCondition**
If retain boot bus width and boot mode conditions.
- **bool pwrBootConfigProtection**
Disable the change of boot configuration register bits from at this point until next power cycle or next H/W reset operation
- **bool premBootConfigProtection**
Disable the change of boot configuration register bits permanently.
- **mmc_boot_partition_wp_t bootPartitionWP**

boot partition write protect configurations

38.3.3 Macro Definition Documentation

38.3.3.1 `#define SDMMC_LOG(format, ...)`

38.3.3.2 `#define READ_MMC_TRANSFER_SPEED_FREQUENCY_UNIT(CSD) (((CSD).transferSpeed) & MMC_TRANSFER_SPEED_FREQUENCY_UNIT_MASK) >> MMC_TRANSFER_SPEED_FREQUENCY_UNIT_SHIFT)`

38.3.3.3 `#define READ_MMC_TRANSFER_SPEED_MULTIPLIER(CSD) (((CSD).transferSpeed) & MMC_TRANSFER_SPEED_MULTIPLIER_MASK) >> MMC_TRANSFER_SPEED_MULTIPLIER_SHIFT)`

38.3.3.4 `#define MMC_EXTENDED_CSD_BYTES (512U)`

38.3.3.5 `#define SD_PRODUCT_NAME_BYTES (5U)`

38.3.3.6 `#define MMC_PRODUCT_NAME_BYTES (6U)`

38.3.3.7 `#define MMC_SWITCH_COMMAND_SET_SHIFT (0U)`

38.3.3.8 `#define MMC_SWITCH_COMMAND_SET_MASK (0x00000007U)`

38.3.4 Enumeration Type Documentation

38.3.4.1 anonymous enum

Enumerator

`kStatus_SDMMC_NotSupportYet` Haven't supported.

`kStatus_SDMMC_TransferFailed` Send command failed.

`kStatus_SDMMC_SetCardBlockSizeFailed` Set block size failed.

`kStatus_SDMMC_HostNotSupport` Host doesn't support.

`kStatus_SDMMC_CardNotSupport` Card doesn't support.

`kStatus_SDMMC_AllSendCidFailed` Send CID failed.

`kStatus_SDMMC_SendRelativeAddressFailed` Send relative address failed.

`kStatus_SDMMC_SendCsdFailed` Send CSD failed.

`kStatus_SDMMC_SelectCardFailed` Select card failed.

`kStatus_SDMMC_SendScrFailed` Send SCR failed.

`kStatus_SDMMC_SetDataBusWidthFailed` Set bus width failed.

`kStatus_SDMMC_GoIdleFailed` Go idle failed.

`kStatus_SDMMC_HandShakeOperationConditionFailed` Send Operation Condition failed.

`kStatus_SDMMC_SendApplicationCommandFailed` Send application command failed.

kStatus_SDMMC_SwitchFailed Switch command failed.
kStatus_SDMMC_StopTransmissionFailed Stop transmission failed.
kStatus_SDMMC_WaitWriteCompleteFailed Wait write complete failed.
kStatus_SDMMC_SetBlockCountFailed Set block count failed.
kStatus_SDMMC_SetRelativeAddressFailed Set relative address failed.
kStatus_SDMMC_SwitchBusTimingFailed Switch high speed failed.
kStatus_SDMMC_SendExtendedCsdFailed Send EXT_CSD failed.
kStatus_SDMMC_ConfigureBootFailed Configure boot failed.
kStatus_SDMMC_ConfigureExtendedCsdFailed Configure EXT_CSD failed.
kStatus_SDMMC_EnableHighCapacityEraseFailed Enable high capacity erase failed.
kStatus_SDMMC_SendTestPatternFailed Send test pattern failed.
kStatus_SDMMC_ReceiveTestPatternFailed Receive test pattern failed.
kStatus_SDMMC_SDIO_ResponseError sdio response error
kStatus_SDMMC_SDIO_InvalidArgument sdio invalid argument response error
kStatus_SDMMC_SDIO_SendOperationConditionFail sdio send operation condition fail
kStatus_SDMMC_InvalidVoltage invalid voltage
kStatus_SDMMC_SDIO_SwitchHighSpeedFail switch to high speed fail
kStatus_SDMMC_SDIO_ReadCISFail read CIS fail
kStatus_SDMMC_SDIO_InvalidCard invalid SDIO card
kStatus_SDMMC_TuningFail tuning fail
kStatus_SDMMC_SwitchVoltageFail switch voltage fail
kStatus_SDMMC_SwitchVoltage18VFail33VSuccess switch voltage fail
kStatus_SDMMC_ReTuningRequest retuning request
kStatus_SDMMC_SetDriverStrengthFail set driver strength fail
kStatus_SDMMC_SetPowerClassFail set power class fail
kStatus_SDMMC_HostNotReady host controller not ready
kStatus_SDMMC_CardDetectFailed card detect failed
kStatus_SDMMC_AuSizeNotSetProperly AU size not set properly.
kStatus_SDMMC_PollingCardIdleFailed polling card idle status failed
kStatus_SDMMC_DeselectCardFailed deselect card failed
kStatus_SDMMC_CardStatusIdle card idle
kStatus_SDMMC_CardStatusBusy card busy
kStatus_SDMMC_CardInitFailed card init failed

38.3.4.2 anonymous enum

Enumerator

<i>kSDMMC_SignalLineCmd</i>	cmd line
<i>kSDMMC_SignalLineData0</i>	data line
<i>kSDMMC_SignalLineData1</i>	data line
<i>kSDMMC_SignalLineData2</i>	data line
<i>kSDMMC_SignalLineData3</i>	data line
<i>kSDMMC_SignalLineData4</i>	data line
<i>kSDMMC_SignalLineData5</i>	data line

kSDMMC_SignalLineData6 data line
kSDMMC_SignalLineData7 data line

38.3.4.3 enum sdmmc_operation_voltage_t

Enumerator

kSDMMC_OperationVoltageNone indicate current voltage setting is not setting by suser
kSDMMC_OperationVoltage330V card operation voltage around 3.3v
kSDMMC_OperationVoltage300V card operation voltage around 3.0v
kSDMMC_OperationVoltage180V card operation voltage around 1.8v

38.3.4.4 anonymous enum

Enumerator

kSDMMC_BusWidth1Bit card bus 1 width
kSDMMC_BusWidth4Bit card bus 4 width
kSDMMC_BusWidth8Bit card bus 8 width

38.3.4.5 anonymous enum

Enumerator

kSDMMC_Support8BitWidth 8 bit data width capability

38.3.4.6 anonymous enum

Enumerator

kSDMMC_DataPacketFormatLSBFirst usual data packet format LSB first, MSB last
kSDMMC_DataPacketFormatMSBFirst Wide width data packet format MSB first, LSB last.

38.3.4.7 enum sd_detect_card_type_t

Enumerator

kSD_DetectCardByGpioCD sd card detect by CD pin through GPIO
kSD_DetectCardByHostCD sd card detect by CD pin through host
kSD_DetectCardByHostDATA3 sd card detect by DAT3 pin through host

38.3.4.8 anonymous enum

Enumerator

kSD_Inserted card is inserted

kSD_Removed card is removed

38.3.4.9 anonymous enum

Enumerator

kSD_DAT3PullDown data3 pull down

kSD_DAT3PullUp data3 pull up

38.3.4.10 enum sd_io_voltage_ctrl_type_t

Enumerator

kSD_IOVoltageCtrlNotSupport io voltage control not support

kSD_IOVoltageCtrlByHost io voltage control by host

kSD_IOVoltageCtrlByGpio io voltage control by gpio

38.3.4.11 anonymous enum

Enumerator

kSDMMC_R1OutOfRangeFlag Out of range status bit.

kSDMMC_R1AddressErrorFlag Address error status bit.

kSDMMC_R1BlockLengthErrorFlag Block length error status bit.

kSDMMC_R1EraseSequenceErrorFlag Erase sequence error status bit.

kSDMMC_R1EraseParameterErrorFlag Erase parameter error status bit.

kSDMMC_R1WriteProtectViolationFlag Write protection violation status bit.

kSDMMC_R1CardIsLockedFlag Card locked status bit.

kSDMMC_R1LockUnlockFailedFlag lock/unlock error status bit

kSDMMC_R1CommandCrcErrorFlag CRC error status bit.

kSDMMC_R1IllegalCommandFlag Illegal command status bit.

kSDMMC_R1CardEccFailedFlag Card ecc error status bit.

kSDMMC_R1CardControllerErrorFlag Internal card controller error status bit.

kSDMMC_R1ErrorFlag A general or an unknown error status bit.

kSDMMC_R1CidCsdOverwriteFlag Cid/csd overwrite status bit.

kSDMMC_R1WriteProtectEraseSkipFlag Write protection erase skip status bit.

kSDMMC_R1CardEccDisabledFlag Card ecc disabled status bit.

kSDMMC_R1EraseResetFlag Erase reset status bit.

kSDMMC_R1ReadyForDataFlag Ready for data status bit.

kSDMMC_R1SwitchErrorFlag Switch error status bit.

kSDMMC_R1ApplicationCommandFlag Application command enabled status bit.

kSDMMC_R1AuthenticationSequenceErrorFlag error in the sequence of authentication process

38.3.4.12 enum sdmmc_r1_current_state_t

Enumerator

kSDMMC_R1StateIdle R1: current state: idle.

kSDMMC_R1StateReady R1: current state: ready.

kSDMMC_R1StateIdentify R1: current state: identification.

kSDMMC_R1StateStandby R1: current state: standby.

kSDMMC_R1StateTransfer R1: current state: transfer.

kSDMMC_R1StateSendData R1: current state: sending data.

kSDMMC_R1StateReceiveData R1: current state: receiving data.

kSDMMC_R1StateProgram R1: current state: programming.

kSDMMC_R1StateDisconnect R1: current state: disconnect.

38.3.4.13 anonymous enum

Enumerator

kSDSPI_R1InIdleStateFlag In idle state.

kSDSPI_R1EraseResetFlag Erase reset.

kSDSPI_R1IllegalCommandFlag Illegal command.

kSDSPI_R1CommandCrcErrorFlag Com crc error.

kSDSPI_R1EraseSequenceErrorFlag Erase sequence error.

kSDSPI_R1AddressErrorFlag Address error.

kSDSPI_R1ParameterErrorFlag Parameter error.

38.3.4.14 anonymous enum

Enumerator

kSDSPI_R2CardLockedFlag Card is locked.

kSDSPI_R2WriteProtectEraseSkip Write protect erase skip.

kSDSPI_R2LockUnlockFailed Lock/unlock command failed.

kSDSPI_R2ErrorFlag Unknown error.

kSDSPI_R2CardControllerErrorFlag Card controller error.

kSDSPI_R2CardEccFailedFlag Card ecc failed.

kSDSPI_R2WriteProtectViolationFlag Write protect violation.

kSDSPI_R2EraseParameterErrorFlag Erase parameter error.

kSDSPI_R2OutOfRangeFlag Out of range.

kSDSPI_R2CsdOverwriteFlag CSD overwrite.

38.3.4.15 anonymous enum

Enumerator

- kSDSPI_DataErrorTokenError* Data error.
- kSDSPI_DataErrorTokenCardControllerError* Card controller error.
- kSDSPI_DataErrorTokenCardEccFailed* Card ecc error.
- kSDSPI_DataErrorTokenOutOfRange* Out of range.

38.3.4.16 enum sdspi_data_token_t

Enumerator

- kSDSPI_DataTokenBlockRead* Single block read, multiple block read.
- kSDSPI_DataTokenSingleBlockWrite* Single block write.
- kSDSPI_DataTokenMultipleBlockWrite* Multiple block write.
- kSDSPI_DataTokenStopTransfer* Stop transmission.

38.3.4.17 enum sdspi_data_response_token_t

Enumerator

- kSDSPI_DataResponseTokenAccepted* Data accepted.
- kSDSPI_DataResponseTokenCrcError* Data rejected due to CRC error.
- kSDSPI_DataResponseTokenWriteError* Data rejected due to write error.

38.3.4.18 enum sd_command_t

Enumerator

- kSD_SendRelativeAddress* Send Relative Address.
- kSD_Switch* Switch Function.
- kSD_SendInterfaceCondition* Send Interface Condition.
- kSD_VoltageSwitch* Voltage Switch.
- kSD_SpeedClassControl* Speed Class control.
- kSD_EraseWriteBlockStart* Write Block Start.
- kSD_EraseWriteBlockEnd* Write Block End.
- kSD_SendTuningBlock* Send Tuning Block.

38.3.4.19 enum sdspi_command_t

Enumerator

- kSDSPI_CommandCrc* Command crc protection on/off.

38.3.4.20 enum sd_application_command_t

Enumerator

kSD_ApplicationSetBusWidth Set Bus Width.
kSD_ApplicationStatus Send SD status.
kSD_ApplicationSendNumberWriteBlocks Send Number Of Written Blocks.
kSD_ApplicationSetWriteBlockEraseCount Set Write Block Erase Count.
kSD_ApplicationSendOperationCondition Send Operation Condition.
kSD_ApplicationSetClearCardDetect Set Connnect/Disconnect pull up on detect pin.
kSD_ApplicationSendScr Send Scr.

38.3.4.21 anonymous enum

Enumerator

kSDMMC_CommandClassBasic Card command class 0.
kSDMMC_CommandClassBlockRead Card command class 2.
kSDMMC_CommandClassBlockWrite Card command class 4.
kSDMMC_CommandClassErase Card command class 5.
kSDMMC_CommandClassWriteProtect Card command class 6.
kSDMMC_CommandClassLockCard Card command class 7.
kSDMMC_CommandClassApplicationSpecific Card command class 8.
kSDMMC_CommandClassInputOutputMode Card command class 9.
kSDMMC_CommandClassSwitch Card command class 10.

38.3.4.22 anonymous enum

Enumerator

kSD_OcrPowerUpBusyFlag Power up busy status.
kSD_OcrHostCapacitySupportFlag Card capacity status.
kSD_OcrCardCapacitySupportFlag Card capacity status.
kSD_OcrSwitch18RequestFlag Switch to 1.8V request.
kSD_OcrSwitch18AcceptFlag Switch to 1.8V accepted.
kSD_OcrVdd27_28Flag VDD 2.7-2.8.
kSD_OcrVdd28_29Flag VDD 2.8-2.9.
kSD_OcrVdd29_30Flag VDD 2.9-3.0.
kSD_OcrVdd30_31Flag VDD 2.9-3.0.
kSD_OcrVdd31_32Flag VDD 3.0-3.1.
kSD_OcrVdd32_33Flag VDD 3.1-3.2.
kSD_OcrVdd33_34Flag VDD 3.2-3.3.
kSD_OcrVdd34_35Flag VDD 3.3-3.4.
kSD_OcrVdd35_36Flag VDD 3.4-3.5.

38.3.4.23 anonymous enum

Enumerator

kSD_SpecificationVersion1_0 SD card version 1.0-1.01.

kSD_SpecificationVersion1_1 SD card version 1.10.

kSD_SpecificationVersion2_0 SD card version 2.00.

kSD_SpecificationVersion3_0 SD card version 3.0.

38.3.4.24 enum sd_switch_mode_t

Enumerator

kSD_SwitchCheck SD switch mode 0: check function.

kSD_SwitchSet SD switch mode 1: set function.

38.3.4.25 anonymous enum

Enumerator

kSD_CsdReadBlockPartialFlag Partial blocks for read allowed [79:79].

kSD_CsdWriteBlockMisalignFlag Write block misalignment [78:78].

kSD_CsdReadBlockMisalignFlag Read block misalignment [77:77].

kSD_CsdDsrImplementedFlag DSR implemented [76:76].

kSD_CsdEraseBlockEnabledFlag Erase single block enabled [46:46].

kSD_CsdWriteProtectGroupEnabledFlag Write protect group enabled [31:31].

kSD_CsdWriteBlockPartialFlag Partial blocks for write allowed [21:21].

kSD_CsdFileFormatGroupFlag File format group [15:15].

kSD_CsdCopyFlag Copy flag [14:14].

kSD_CsdPermanentWriteProtectFlag Permanent write protection [13:13].

kSD_CsdTemporaryWriteProtectFlag Temporary write protection [12:12].

38.3.4.26 anonymous enum

Enumerator

kSD_ScrDataStatusAfterErase Data status after erases [55:55].

kSD_ScrSdSpecification3 Specification version 3.00 or higher [47:47].

38.3.4.27 anonymous enum

Enumerator

kSD_FunctionSDR12Default SDR12 mode & default.

kSD_FunctionSDR25HighSpeed SDR25 & high speed.

kSD_FunctionSDR50 SDR50 mode.
kSD_FunctionSDR104 SDR104 mode.
kSD_FunctionDDR50 DDR50 mode.

38.3.4.28 anonymous enum

Enumerator

kSD_GroupTimingMode access mode group
kSD_GroupCommandSystem command system group
kSD_GroupDriverStrength driver strength group
kSD_GroupCurrentLimit current limit group

38.3.4.29 enum sd_timing_mode_t

Enumerator

kSD_TimingSDR12DefaultMode Identification mode & SDR12.
kSD_TimingSDR25HighSpeedMode High speed mode & SDR25.
kSD_TimingSDR50Mode SDR50 mode.
kSD_TimingSDR104Mode SDR104 mode.
kSD_TimingDDR50Mode DDR50 mode.

38.3.4.30 enum sd_driver_strength_t

Enumerator

kSD_DriverStrengthTypeB default driver strength
kSD_DriverStrengthTypeA driver strength TYPE A
kSD_DriverStrengthTypeC driver strength TYPE C
kSD_DriverStrengthTypeD driver strength TYPE D

38.3.4.31 enum sd_max_current_t

Enumerator

kSD_CurrentLimit200MA default current limit
kSD_CurrentLimit400MA current limit to 400MA
kSD_CurrentLimit600MA current limit to 600MA
kSD_CurrentLimit800MA current limit to 800MA

38.3.4.32 enum sdmmc_command_t

Enumerator

kSDMMC_GoIdleState Go Idle State.
kSDMMC_AllSendCid All Send CID.
kSDMMC_SetDsr Set DSR.
kSDMMC_SelectCard Select Card.
kSDMMC_SendCsd Send CSD.
kSDMMC_SendCid Send CID.
kSDMMC_StopTransmission Stop Transmission.
kSDMMC_SendStatus Send Status.
kSDMMC_GoInactiveState Go Inactive State.
kSDMMC_SetBlockLength Set Block Length.
kSDMMC_ReadSingleBlock Read Single Block.
kSDMMC_ReadMultipleBlock Read Multiple Block.
kSDMMC_SetBlockCount Set Block Count.
kSDMMC_WriteSingleBlock Write Single Block.
kSDMMC_WriteMultipleBlock Write Multiple Block.
kSDMMC_ProgramCsd Program CSD.
kSDMMC_SetWriteProtect Set Write Protect.
kSDMMC_ClearWriteProtect Clear Write Protect.
kSDMMC_SendWriteProtect Send Write Protect.
kSDMMC_Erase Erase.
kSDMMC_LockUnlock Lock Unlock.
kSDMMC_ApplicationCommand Send Application Command.
kSDMMC_GeneralCommand General Purpose Command.
kSDMMC_ReadOcr Read OCR.

38.3.4.33 anonymous enum

Enumerator

kSDIO_RegCCCRSdioVer CCCR & SDIO version.
kSDIO_RegSDVersion SD version.
kSDIO_RegIOEnable io enable register
kSDIO_RegIOReady io ready register
kSDIO_RegIOIntEnable io interrupt enable register
kSDIO_RegIOIntPending io interrupt pending register
kSDIO_RegIOAbort io abort register
kSDIO_RegBusInterface bus interface register
kSDIO_RegCardCapability card capability register
kSDIO_RegCommonCISPointer common CIS pointer register
kSDIO_RegBusSuspend bus suspend register
kSDIO_RegFunctionSelect function select register
kSDIO_RegExecutionFlag execution flag register

kSDIO_RegReadyFlag ready flag register
kSDIO_RegFN0BlockSizeLow FN0 block size register.
kSDIO_RegFN0BlockSizeHigh FN0 block size register.
kSDIO_RegPowerControl power control register
kSDIO_RegBusSpeed bus speed register
kSDIO_RegUHSITimingSupport UHS-I timing support register.
kSDIO_RegDriverStrength Driver strength register.
kSDIO_RegInterruptExtension Interrupt extension register.

38.3.4.34 enum sdio_command_t

Enumerator

kSDIO_SendRelativeAddress send relative address
kSDIO_SendOperationCondition send operation condition
kSDIO_SendInterfaceCondition send interface condition
kSDIO_RWIODirect read/write IO direct command
kSDIO_RWIOExtended read/write IO extended command

38.3.4.35 enum sdio_func_num_t

Enumerator

kSDIO_FunctionNum0 sdio function0
kSDIO_FunctionNum1 sdio function1
kSDIO_FunctionNum2 sdio function2
kSDIO_FunctionNum3 sdio function3
kSDIO_FunctionNum4 sdio function4
kSDIO_FunctionNum5 sdio function5
kSDIO_FunctionNum6 sdio function6
kSDIO_FunctionNum7 sdio function7
kSDIO_FunctionMemory for combo card

38.3.4.36 anonymous enum

Enumerator

kSDIO_StatusCmdCRCError the CRC check of the previous cmd fail
kSDIO_StatusIllegalCmd cmd illegal for the card state
kSDIO_StatusR6Error special for R6 error status
kSDIO_StatusError A general or an unknown error occurred.
kSDIO_StatusFunctionNumError invalid function error
kSDIO_StatusOutOfRange cmd argument was out of the allowed range

38.3.4.37 anonymous enum

Enumerator

<i>kSDIO_OcrPowerUpBusyFlag</i>	Power up busy status.
<i>kSDIO_OcrIONumber</i>	number of IO function
<i>kSDIO_OcrMemPresent</i>	memory present flag
<i>kSDIO_OcrVdd20_21Flag</i>	VDD 2.0-2.1.
<i>kSDIO_OcrVdd21_22Flag</i>	VDD 2.1-2.2.
<i>kSDIO_OcrVdd22_23Flag</i>	VDD 2.2-2.3.
<i>kSDIO_OcrVdd23_24Flag</i>	VDD 2.3-2.4.
<i>kSDIO_OcrVdd24_25Flag</i>	VDD 2.4-2.5.
<i>kSDIO_OcrVdd25_26Flag</i>	VDD 2.5-2.6.
<i>kSDIO_OcrVdd26_27Flag</i>	VDD 2.6-2.7.
<i>kSDIO_OcrVdd27_28Flag</i>	VDD 2.7-2.8.
<i>kSDIO_OcrVdd28_29Flag</i>	VDD 2.8-2.9.
<i>kSDIO_OcrVdd29_30Flag</i>	VDD 2.9-3.0.
<i>kSDIO_OcrVdd30_31Flag</i>	VDD 2.9-3.0.
<i>kSDIO_OcrVdd31_32Flag</i>	VDD 3.0-3.1.
<i>kSDIO_OcrVdd32_33Flag</i>	VDD 3.1-3.2.
<i>kSDIO_OcrVdd33_34Flag</i>	VDD 3.2-3.3.
<i>kSDIO_OcrVdd34_35Flag</i>	VDD 3.3-3.4.
<i>kSDIO_OcrVdd35_36Flag</i>	VDD 3.4-3.5.

38.3.4.38 anonymous enum

Enumerator

<i>kSDIO_CCCRSupportDirectCmdDuringDataTrans</i>	support direct cmd during data transfer
<i>kSDIO_CCCRSupportMultiBlock</i>	support multi block mode
<i>kSDIO_CCCRSupportReadWait</i>	support read wait
<i>kSDIO_CCCRSupportSuspendResume</i>	support suspend resume
<i>kSDIO_CCCRSupportIntDuring4BitDataTrans</i>	support interrupt during 4-bit data transfer
<i>kSDIO_CCCRSupportLowSpeed1Bit</i>	support low speed 1bit mode
<i>kSDIO_CCCRSupportLowSpeed4Bit</i>	support low speed 4bit mode
<i>kSDIO_CCCRSupportMasterPowerControl</i>	support master power control
<i>kSDIO_CCCRSupportHighSpeed</i>	support high speed
<i>kSDIO_CCCRSupportContinuousSPIInt</i>	support continuous SPI interrupt

38.3.4.39 anonymous enum

Enumerator

<i>kSDIO_FBRSupportCSA</i>	function support CSA
<i>kSDIO_FBRSupportPowerSelection</i>	function support power selection

38.3.4.40 enum sdio_bus_width_t

Enumerator

kSDIO_DataBus1Bit 1 bit bus mode
kSDIO_DataBus4Bit 4 bit bus mode
kSDIO_DataBus8Bit 8 bit bus mode

38.3.4.41 enum mmc_command_t

Enumerator

kMMC_SendOperationCondition Send Operation Condition.
kMMC_SetRelativeAddress Set Relative Address.
kMMC_SleepAwake Sleep Awake.
kMMC_Switch Switch.
kMMC_SendExtendedCsd Send EXT_CSD.
kMMC_ReadDataUntilStop Read Data Until Stop.
kMMC_BusTestRead Test Read.
kMMC_SendingBusTest test bus width cmd
kMMC_WriteDataUntilStop Write Data Until Stop.
kMMC_SendTuningBlock MMC sending tuning block.
kMMC_ProgramCid Program CID.
kMMC_EraseGroupStart Erase Group Start.
kMMC_EraseGroupEnd Erase Group End.
kMMC_FastInputOutput Fast IO.
kMMC_GoInterruptState Go interrupt State.

38.3.4.42 enum mmc_classified_voltage_t

Enumerator

kMMC_ClassifiedVoltageHigh High-voltage MMC card.
kMMC_ClassifiedVoltageDual Dual-voltage MMC card.

38.3.4.43 enum mmc_classified_density_t

Enumerator

kMMC_ClassifiedDensityWithin2GB Density byte is less than or equal 2GB.

38.3.4.44 enum mmc_access_mode_t

Enumerator

kMMC_AccessModeByte The card should be accessed as byte.

kMMC_AccessModeSector The card should be accessed as sector.

38.3.4.45 enum mmc_voltage_window_t

Enumerator

kMMC_VoltageWindowNone voltage window is not define by user

kMMC_VoltageWindow120 Voltage window is 1.20V.

kMMC_VoltageWindow170to195 Voltage window is 1.70V to 1.95V.

kMMC_VoltageWindows270to360 Voltage window is 2.70V to 3.60V.

38.3.4.46 enum mmc_csd_structure_version_t

Enumerator

kMMC_CsdStrucureVersion10 CSD version No. 1.0

kMMC_CsdStrucureVersion11 CSD version No. 1.1

kMMC_CsdStrucureVersion12 CSD version No. 1.2

kMMC_CsdStrucureVersionInExtcsd Version coded in Extended CSD.

38.3.4.47 enum mmc_specification_version_t

Enumerator

kMMC_SpecificationVersion0 Allocated by MMCA.

kMMC_SpecificationVersion1 Allocated by MMCA.

kMMC_SpecificationVersion2 Allocated by MMCA.

kMMC_SpecificationVersion3 Allocated by MMCA.

kMMC_SpecificationVersion4 Version 4.1/4.2/4.3/4.41-4.5-4.51-5.0.

38.3.4.48 anonymous enum

Enumerator

kMMC_ExtendedCsdRevision10 Revision 1.0.

kMMC_ExtendedCsdRevision11 Revision 1.1.

kMMC_ExtendedCsdRevision12 Revision 1.2.

kMMC_ExtendedCsdRevision13 Revision 1.3 MMC4.3.

kMMC_ExtendedCsdRevision14 Revision 1.4 obsolete.

- kMMC_ExtendedCsdRevision15* Revision 1.5 MMC4.41.
- kMMC_ExtendedCsdRevision16* Revision 1.6 MMC4.5.
- kMMC_ExtendedCsdRevision17* Revision 1.7 MMC5.0.

38.3.4.49 enum mmc_command_set_t

Enumerator

- kMMC_CommandSetStandard* Standard MMC.
- kMMC_CommandSet1* Command set 1.
- kMMC_CommandSet2* Command set 2.
- kMMC_CommandSet3* Command set 3.
- kMMC_CommandSet4* Command set 4.

38.3.4.50 anonymous enum

Enumerator

- kMMC_SupportAlternateBoot* support alternative boot mode
- kMMC_SupportDDRBoot* support DDR boot mode
- kMMC_SupportHighSpeedBoot* support high speed boot mode

38.3.4.51 enum mmc_high_speed_timing_t

Enumerator

- kMMC_HighSpeedTimingNone* MMC card using none high-speed timing.
- kMMC_HighSpeedTiming* MMC card using high-speed timing.
- kMMC_HighSpeed200Timing* MMC card high speed 200 timing.
- kMMC_HighSpeed400Timing* MMC card high speed 400 timing.
- kMMC_EnhanceHighSpeed400Timing* MMC card high speed 400 timing.

38.3.4.52 enum mmc_data_bus_width_t

Enumerator

- kMMC_DataBusWidth1bit* MMC data bus width is 1 bit.
- kMMC_DataBusWidth4bit* MMC data bus width is 4 bits.
- kMMC_DataBusWidth8bit* MMC data bus width is 8 bits.
- kMMC_DataBusWidth4bitDDR* MMC data bus width is 4 bits ddr.
- kMMC_DataBusWidth8bitDDR* MMC data bus width is 8 bits ddr.
- kMMC_DataBusWidth8bitDDRSTROBE* MMC data bus width is 8 bits ddr strobe mode.

38.3.4.53 enum mmc_boot_partition_enable_t

Enumerator

kMMC_BootPartitionEnableNot Device not boot enabled (default)
kMMC_BootPartitionEnablePartition1 Boot partition 1 enabled for boot.
kMMC_BootPartitionEnablePartition2 Boot partition 2 enabled for boot.
kMMC_BootPartitionEnableUserAera User area enabled for boot.

38.3.4.54 enum mmc_boot_timing_mode_t

Enumerator

kMMC_BootModeSDRWithDefaultTiming boot mode single data rate with backward compatible timings
kMMC_BootModeSDRWithHighSpeedTiming boot mode single data rate with high speed timing
kMMC_BootModeDDRTiming boot mode dual date rate

38.3.4.55 enum mmc_boot_partition_wp_t

Enumerator

kMMC_BootPartitionWPDisable boot partition write protection disable
kMMC_BootPartitionPwrWPToBothPartition power on period write protection apply to both boot partitions
kMMC_BootPartitionPermWPToBothPartition permanent write protection apply to both boot partitions
kMMC_BootPartitionPwrWPToPartition1 power on period write protection apply to partition1
kMMC_BootPartitionPwrWPToPartition2 power on period write protection apply to partition2
kMMC_BootPartitionPermWPToPartition1 permanent write protection apply to partition1
kMMC_BootPartitionPermWPToPartition2 permanent write protection apply to partition2
kMMC_BootPartitionPermWPToPartition1PwrWPToPartition2 permanent write protection apply to partition1, power on period write protection apply to partition2
kMMC_BootPartitionPermWPToPartition2PwrWPToPartition1 permanent write protection apply to partition2, power on period write protection apply to partition1

38.3.4.56 anonymous enum

Enumerator

kMMC_BootPartitionNotProtected boot partition not protected
kMMC_BootPartitionPwrProtected boot partition is power on period write protected
kMMC_BootPartitionPermProtected boot partition is permanently protected

38.3.4.57 enum mmc_access_partition_t

Enumerator

kMMC_AccessPartitionUserAera No access to boot partition (default), normal partition.
kMMC_AccessPartitionBoot1 Read/Write boot partition 1.
kMMC_AccessPartitionBoot2 Read/Write boot partition 2.
kMMC_AccessRPMB Replay protected mem block.
kMMC_AccessGeneralPurposePartition1 access to general purpose partition 1
kMMC_AccessGeneralPurposePartition2 access to general purpose partition 2
kMMC_AccessGeneralPurposePartition3 access to general purpose partition 3
kMMC_AccessGeneralPurposePartition4 access to general purpose partition 4

38.3.4.58 anonymous enum

Enumerator

kMMC_CsdReadBlockPartialFlag Partial blocks for read allowed.
kMMC_CsdWriteBlockMisalignFlag Write block misalignment.
kMMC_CsdReadBlockMisalignFlag Read block misalignment.
kMMC_CsdDsrImplementedFlag DSR implemented.
kMMC_CsdWriteProtectGroupEnabledFlag Write protect group enabled.
kMMC_CsdWriteBlockPartialFlag Partial blocks for write allowed.
kMMC_ContentProtectApplicationFlag Content protect application.
kMMC_CsdFileFormatGroupFlag File format group.
kMMC_CsdCopyFlag Copy flag.
kMMC_CsdPermanentWriteProtectFlag Permanent write protection.
kMMC_CsdTemporaryWriteProtectFlag Temporary write protection.

38.3.4.59 enum mmc_extended_csd_access_mode_t

Enumerator

kMMC_ExtendedCsdAccessModeCommandSet Command set related setting.
kMMC_ExtendedCsdAccessModeSetBits Set bits in specific byte in Extended CSD.
kMMC_ExtendedCsdAccessModeClearBits Clear bits in specific byte in Extended CSD.
kMMC_ExtendedCsdAccessModeWriteBits Write a value to specific byte in Extended CSD.

38.3.4.60 enum mmc_extended_csd_index_t

Enumerator

kMMC_ExtendedCsdIndexFlushCache flush cache
kMMC_ExtendedCsdIndexCacheControl cache control

kMMC_ExtendedCsdIndexBootPartitionWP Boot partition write protect.
kMMC_ExtendedCsdIndexEraseGroupDefinition Erase Group Def.
kMMC_ExtendedCsdIndexBootBusConditions Boot Bus conditions.
kMMC_ExtendedCsdIndexBootConfigWP Boot config write protect.
kMMC_ExtendedCsdIndexPartitionConfig Partition Config, before BOOT_CONFIG.
kMMC_ExtendedCsdIndexBusWidth Bus Width.
kMMC_ExtendedCsdIndexHighSpeedTiming High-speed Timing.
kMMC_ExtendedCsdIndexPowerClass Power Class.
kMMC_ExtendedCsdIndexCommandSet Command Set.

38.3.4.61 anonymous enum

Enumerator

kMMC_DriverStrength0 Driver type0 ,nominal impedance 50ohm.
kMMC_DriverStrength1 Driver type1 ,nominal impedance 33ohm.
kMMC_DriverStrength2 Driver type2 ,nominal impedance 66ohm.
kMMC_DriverStrength3 Driver type3 ,nominal impedance 100ohm.
kMMC_DriverStrength4 Driver type4 ,nominal impedance 40ohm.

38.3.4.62 enum mmc_extended_csd_flags_t

Enumerator

kMMC_ExtCsdExtPartitionSupport partitioning support[160]
kMMC_ExtCsdEnhancePartitionSupport partitioning support[160]
kMMC_ExtCsdPartitioningSupport partitioning support[160]
kMMC_ExtCsdPrgCIDCSDInDDRModeSupport CMD26 and CMD27 are support dual data rate [130].
kMMC_ExtCsdBKOpsSupport background operation feature support [502]
kMMC_ExtCsdDataTagSupport data tag support[499]
kMMC_ExtCsdModeOperationCodeSupport mode operation code support[493]

38.3.4.63 enum mmc_boot_mode_t

Enumerator

kMMC_BootModeNormal Normal boot.
kMMC_BootModeAlternative Alternative boot.

38.3.5 Function Documentation

38.3.5.1 **status_t SDMMC_SelectCard (*sdmmchost_t * host, uint32_t relativeAddress, bool isSelected*)**

Parameters

<i>host</i>	host handler.
<i>relativeAddress</i>	Relative address.
<i>isSelected</i>	True to put card into transfer state.

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	Transfer failed.
<i>kStatus_Success</i>	Operate successfully.

38.3.5.2 status_t SDMMC_SendApplicationCommand (*sdmmchost_t * host, uint32_t relativeAddress*)

Parameters

<i>host</i>	host handler.
<i>relativeAddress</i>	Card relative address.

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_Card-NotSupport</i>	Card doesn't support.
<i>kStatus_Success</i>	Operate successfully.

38.3.5.3 status_t SDMMC_SetBlockCount (*sdmmchost_t * host, uint32_t blockCount*)

Parameters

<i>host</i>	host handler.
<i>blockCount</i>	Block count.

Return values

<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_Success</i>	Operate successfully.

38.3.5.4 status_t SDMMC_GoIdle (**sdmmchost_t * host**)

Parameters

<i>host</i>	host handler.
-------------	---------------

Return values

<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_Success</i>	Operate successfully.

38.3.5.5 status_t SDMMC_SetBlockSize (**sdmmchost_t * host, uint32_t blockSize**)

Parameters

<i>host</i>	host handler.
<i>blockSize</i>	Block size.

Return values

<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_Success</i>	Operate successfully.

38.3.5.6 status_t SDMMC_SetCardInactive (**sdmmchost_t * host**)

Parameters

<i>host</i>	host handler.
-------------	---------------

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	Transfer failed.
<i>kStatus_Success</i>	Operate successfully.

Chapter 39

SPI based Secure Digital Card (SDSPI)

39.1 Overview

The MCUXpresso SDK provides a driver to access the Secure Digital Card based on the SPI driver.

Function groups

SDSPI Function

This function group implements the SD card functional API in the SPI mode.

Typical use case

SDSPI Operation

```
/* SPI_Init(). */

/* Register the SDSPI driver callback. */

/* Initializes card. */
if (kStatus_Success != SDSPI_Init(card))
{
    SDSPI_Deinit(card)
    return;
}

/* Read/Write card */
memset(g_testWriteBuffer, 0x17U, sizeof(g_testWriteBuffer));

while (true)
{
    memset(g_testReadBuffer, 0U, sizeof(g_testReadBuffer));

    SDSPI_WriteBlocks(card, g_testWriteBuffer, TEST_START_BLOCK, TEST_BLOCK_COUNT);

    SDSPI_ReadBlocks(card, g_testReadBuffer, TEST_START_BLOCK, TEST_BLOCK_COUNT);

    if (memcmp(g_testReadBuffer, g_testReadBuffer, sizeof(g_testWriteBuffer)))
    {
        break;
    }
}
```

Data Structures

- struct `sdspi_host_t`
SDSPI host state. [More...](#)
- struct `sdspi_card_t`
SD Card Structure. [More...](#)

Macros

- `#define FSL_SDSPI_DRIVER_VERSION (MAKE_VERSION(2U, 2U, 1U)) /*2.2.1*/`
Driver version.
- `#define FSL_SDSPI_DEFAULT_BLOCK_SIZE (512U)`
Default block size.
- `#define DSPI_DUMMY_DATA (0xFFU)`
Dummy byte define, 0xFF should be defined as the dummy data.
- `#define SDSPI_CARD_CRC_PROTECTION_ENABLE 0U`
This macro is used to enable or disable the CRC protection for SD card command.

Enumerations

- enum {

`kStatus_SDSPI_SetFrequencyFailed = MAKE_STATUS(kStatusGroup_SDSPI, 0U),`
`kStatus_SDSPI_ExchangeFailed = MAKE_STATUS(kStatusGroup_SDSPI, 1U),`
`kStatus_SDSPI_WaitReadyFailed = MAKE_STATUS(kStatusGroup_SDSPI, 2U),`
`kStatus_SDSPI_ResponseError = MAKE_STATUS(kStatusGroup_SDSPI, 3U),`
`kStatus_SDSPI_WriteProtected = MAKE_STATUS(kStatusGroup_SDSPI, 4U),`
`kStatus_SDSPI_GoIdleFailed = MAKE_STATUS(kStatusGroup_SDSPI, 5U),`
`kStatus_SDSPI_SendCommandFailed = MAKE_STATUS(kStatusGroup_SDSPI, 6U),`
`kStatus_SDSPI_ReadFailed = MAKE_STATUS(kStatusGroup_SDSPI, 7U),`
`kStatus_SDSPI_WriteFailed = MAKE_STATUS(kStatusGroup_SDSPI, 8U),`
`kStatus_SDSPI_SendInterfaceConditionFailed,`
`kStatus_SDSPI_SendOperationConditionFailed,`
`kStatus_SDSPI_ReadOcrFailed = MAKE_STATUS(kStatusGroup_SDSPI, 11U),`
`kStatus_SDSPI_SetBlockSizeFailed = MAKE_STATUS(kStatusGroup_SDSPI, 12U),`
`kStatus_SDSPI_SendCsdFailed = MAKE_STATUS(kStatusGroup_SDSPI, 13U),`
`kStatus_SDSPI_SendCidFailed = MAKE_STATUS(kStatusGroup_SDSPI, 14U),`
`kStatus_SDSPI_StopTransmissionFailed = MAKE_STATUS(kStatusGroup_SDSPI, 15U),`
`kStatus_SDSPI_SendApplicationCommandFailed,`
`kStatus_SDSPI_InvalidVoltage = MAKE_STATUS(kStatusGroup_SDSPI, 17U),`
`kStatus_SDSPI_SwitchCmdFail = MAKE_STATUS(kStatusGroup_SDSPI, 18U),`
`kStatus_SDSPI_NotSupportYet = MAKE_STATUS(kStatusGroup_SDSPI, 19U) }`

SDSPI API status.
- enum {

`kSDSPI_SupportHighCapacityFlag = (1U << 0U),`
`kSDSPI_SupportSdhcFlag = (1U << 1U),`
`kSDSPI_SupportSdxcFlag = (1U << 2U),`
`kSDSPI_SupportSdscFlag = (1U << 3U) }`

SDSPI card flag.
- enum {

`kSDSPI_ResponseTypeR1 = 0U,`
`kSDSPI_ResponseTypeR1b = 1U,`
`kSDSPI_ResponseTypeR2 = 2U,`
`kSDSPI_ResponseTypeR3 = 3U,`
`kSDSPI_ResponseTypeR7 = 4U }`

- *SDSPI response type.*
- enum {

 kSDSPI_CmdGoIdle = kSDMMC_GoIdleState << 8U | kSDSPI_ResponseTypeDef1,

 kSDSPI_CmdCrc = kSDSPI_CommandCrc << 8U | kSDSPI_ResponseTypeDef1,

 kSDSPI_CmdSendInterfaceCondition }
- *SDSPI command type.*
- enum **sdspi_cs_active_polarity_t** {

 kSDSPI_CsActivePolarityHigh = 0U,

 kSDSPI_CsActivePolarityLow }
- cs active polarity*

SDSPI Function

- **status_t SDSPI_Init (sdspi_card_t *card)**

Initializes the card on a specific SPI instance.
- **void SDSPI_Deinit (sdspi_card_t *card)**

Deinitializes the card.
- **bool SDSPI_CheckReadOnly (sdspi_card_t *card)**

Checks whether the card is write-protected.
- **status_t SDSPI_ReadBlocks (sdspi_card_t *card, uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)**

Reads blocks from the specific card.
- **status_t SDSPI_WriteBlocks (sdspi_card_t *card, uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)**

Writes blocks of data to the specific card.
- **status_t SDSPI_Send_cid (sdspi_card_t *card)**

Send GET-CID command In our sdspi init function, this function is removed for better code size, if information is needed, you can call it after the init function directly.
- **status_t SDSPI_SendPreErase (sdspi_card_t *card, uint32_t blockCount)**

Multiple blocks write pre-erase function.
- **status_t SDSPI_EraseBlocks (sdspi_card_t *card, uint32_t startBlock, uint32_t blockCount)**

Block erase function.
- **status_t SDSPI_SwitchToHighSpeed (sdspi_card_t *card)**

Switch to high speed function.

39.2 Data Structure Documentation

39.2.1 struct sdspi_host_t

Data Fields

- **uint32_t busBaudRate**

Bus baud rate.
- **status_t(* setFrequency)(uint32_t frequency)**

Set frequency of SPI.
- **status_t(* exchange)(uint8_t *in, uint8_t *out, uint32_t size)**

Exchange data over SPI.
- **void(* init)(void)**

SPI initialization.

- void(* **deinit**)(void)
SPI de-initialization.
- void(* **csActivePolarity**)(sdspi_cs_active_polarity_t polarity)
SPI CS active polarity.

39.2.2 struct sdspi_card_t

Define the card structure including the necessary fields to identify and describe the card.

Data Fields

- **sdspi_host_t * host**
Host state information.
- **uint32_t relativeAddress**
Relative address of the card.
- **uint32_t flags**
Flags defined in _sdspi_card_flag.
- **uint8_t internalBuffer [16U]**
internal buffer for card raw register content
- **uint32_t ocr**
Raw OCR content.
- **sd_cid_t cid**
CID.
- **sd_csd_t csd**
CSD.
- **sd_scr_t scr**
SCR.
- **uint32_t blockCount**
Card total block number.
- **uint32_t blockSize**
Card block size.

Field Documentation

(1) **uint32_t sdspi_card_t::flags**

39.3 Macro Definition Documentation

#define FSL_SDSPI_DRIVER_VERSION (MAKE_VERSION(2U, 2U, 1U))
/*2.2.1*/

39.3.2 #define DSPI_DUMMY_DATA (0xFFU)

Dummy data used for Tx if there is no txData.

39.3.3 #define SDSPI_CARD_CRC_PROTECTION_ENABLE OU

The SPI interface is initialized in the CRC off mode by default. However, the RESET command(cmd0) that is used to switch the card to SPI mode, is received by the card while in SD mode and therefore, shall have a valid CRC field, after the card put into SPI mode , CRC check for all command include CMD0 will be done according to CMD59 setting, host can turn CRC option on and off using the CMD59, this command should be called before ACMD41. CMD8 CRC verification is always enabled. The host shall set correct CRC in the argument of CMD8. If CRC check is enabled, then sdspi code size and read/write performance will be lower than CRC off. CRC check is off by default.

39.4 Enumeration Type Documentation

39.4.1 anonymous enum

Enumerator

- kStatus_SDSPI_SetFrequencyFailed* Set frequency failed.
- kStatus_SDSPI_ExchangeFailed* Exchange data on SPI bus failed.
- kStatus_SDSPI_WaitReadyFailed* Wait card ready failed.
- kStatus_SDSPI_ResponseError* Response is error.
- kStatus_SDSPI_WriteProtected* Write protected.
- kStatus_SDSPI_GoIdleFailed* Go idle failed.
- kStatus_SDSPI_SendCommandFailed* Send command failed.
- kStatus_SDSPI_ReadFailed* Read data failed.
- kStatus_SDSPI_WriteFailed* Write data failed.
- kStatus_SDSPI_SendInterfaceConditionFailed* Send interface condition failed.
- kStatus_SDSPI_SendOperationConditionFailed* Send operation condition failed.
- kStatus_SDSPI_ReadOcrFailed* Read OCR failed.
- kStatus_SDSPI_SetBlockSizeFailed* Set block size failed.
- kStatus_SDSPI_SendCsdFailed* Send CSD failed.
- kStatus_SDSPI_SendCidFailed* Send CID failed.
- kStatus_SDSPI_StopTransmissionFailed* Stop transmission failed.
- kStatus_SDSPI_SendApplicationCommandFailed* Send application command failed.
- kStatus_SDSPI_InvalidVoltage* invalid supply voltage
- kStatus_SDSPI_SwitchCmdFail* switch command crc protection on/off
- kStatus_SDSPI_NotSupportYet* not support

39.4.2 anonymous enum

Enumerator

- kSDSPI_SupportHighCapacityFlag* Card is high capacity.
- kSDSPI_SupportSdhcFlag* Card is SDHC.
- kSDSPI_SupportSdxcFlag* Card is SDXC.
- kSDSPI_SupportSdscFlag* Card is SDSC.

39.4.3 anonymous enum

Enumerator

- kSDSPI_Response_TypeR1* Response 1.
- kSDSPI_Response_TypeR1b* Response 1 with busy.
- kSDSPI_Response_TypeR2* Response 2.
- kSDSPI_Response_TypeR3* Response 3.
- kSDSPI_Response_TypeR7* Response 7.

39.4.4 anonymous enum

Enumerator

- kSDSPI_CmdGoIdle* command go idle
- kSDSPI_CmdCrc* command crc protection
- kSDSPI_CmdSendInterfaceCondition* command send interface condition

39.4.5 enum sdspi_cs_active_polarity_t

Enumerator

- kSDSPI_CsActivePolarityHigh* CS active polarity high.
- kSDSPI_CsActivePolarityLow* CS active polarity low.

39.5 Function Documentation

39.5.1 status_t SDSPI_Init (*sdspi_card_t* * *card*)

This function initializes the card on a specific SPI instance.

Parameters

<i>card</i>	Card descriptor
-------------	-----------------

Return values

<i>kStatus_SDSPI_Set-FrequencyFailed</i>	Set frequency failed.
--	-----------------------

<i>kStatus_SDSPI_GoIdleFailed</i>	Go idle failed.
<i>kStatus_SDSPI_SendInterfaceConditionFailed</i>	Send interface condition failed.
<i>kStatus_SDSPI_SendOperationConditionFailed</i>	Send operation condition failed.
<i>kStatus_Timeout</i>	Send command timeout.
<i>kStatus_SDSPI_NotSupportYet</i>	Not support yet.
<i>kStatus_SDSPI_ReadOcrFailed</i>	Read OCR failed.
<i>kStatus_SDSPI_SetBlockSizeFailed</i>	Set block size failed.
<i>kStatus_SDSPI_SendCsdFailed</i>	Send CSD failed.
<i>kStatus_SDSPI_SendCidFailed</i>	Send CID failed.
<i>kStatus_Success</i>	Operate successfully.

39.5.2 void SDSPI_Deinit (*sdspi_card_t * card*)

This function deinitializes the specific card.

Parameters

<i>card</i>	Card descriptor
-------------	-----------------

39.5.3 bool SDSPI_CheckReadOnly (*sdspi_card_t * card*)

This function checks if the card is write-protected via CSD register.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>true</i>	Card is read only.
<i>false</i>	Card isn't read only.

39.5.4 status_t SDSPI_ReadBlocks (*sdspi_card_t * card, uint8_t * buffer, uint32_t startBlock, uint32_t blockCount*)

This function reads blocks from specific card.

Parameters

<i>card</i>	Card descriptor.
<i>buffer</i>	the buffer to hold the data read from card
<i>startBlock</i>	the start block index
<i>blockCount</i>	the number of blocks to read

Return values

<i>kStatus_SDSPI_Send-CommandFailed</i>	Send command failed.
<i>kStatus_SDSPI_Read-Failed</i>	Read data failed.
<i>kStatus_SDSPI_Stop-TransmissionFailed</i>	Stop transmission failed.
<i>kStatus_Success</i>	Operate successfully.

39.5.5 status_t SDSPI_WriteBlocks (*sdspi_card_t * card, uint8_t * buffer, uint32_t startBlock, uint32_t blockCount*)

This function writes blocks to specific card

Parameters

<i>card</i>	Card descriptor.
<i>buffer</i>	the buffer holding the data to be written to the card
<i>startBlock</i>	the start block index
<i>blockCount</i>	the number of blocks to write

Return values

<i>kStatus_SDSPI_WriteProtected</i>	Card is write protected.
<i>kStatus_SDSPI_SendCommandFailed</i>	Send command failed.
<i>kStatus_SDSPI_ResponseError</i>	Response is error.
<i>kStatus_SDSPI_WriteFailed</i>	Write data failed.
<i>kStatus_SDSPI_ExchangeFailed</i>	Exchange data over SPI failed.
<i>kStatus_SDSPI_WaitReadyFailed</i>	Wait card to be ready status failed.
<i>kStatus_Success</i>	Operate successfully.

39.5.6 **status_t SDSPI_SendCid(sdspi_card_t * *card*)**

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDSPI_CommandFailed</i>	Send command failed.
<i>kStatus_SDSPI_ReadFailed</i>	Read data blocks failed.

<i>kStatus_Success</i>	Operate successfully.
------------------------	-----------------------

39.5.7 status_t SDSPI_SendPreErase (*sdspi_card_t * card, uint32_t blockCount*)

This function should be called before SDSPI_WriteBlocks, it is used to set the number of the write blocks to be pre-erased before writing.

Parameters

<i>card</i>	Card descriptor.
<i>blockCount</i>	the block counts to be write.

Return values

<i>kStatus_SDSPI_SendCommandFailed</i>	Send command failed.
<i>kStatus_SDSPI_SendApplicationCommandFailed</i>	
<i>kStatus_SDSPI_ResponseError</i>	
<i>kStatus_Success</i>	Operate successfully.

39.5.8 status_t SDSPI_EraseBlocks (*sdspi_card_t * card, uint32_t startBlock, uint32_t blockCount*)

Parameters

<i>card</i>	Card descriptor.
<i>startBlock</i>	start block address to be erase.
<i>blockCount</i>	the block counts to be erase.

Return values

<i>kStatus_SDSPI_WaitReadyFailed</i>	Wait ready failed.
<i>kStatus_SDSPI_SendCommandFailed</i>	Send command failed.
<i>kStatus_Success</i>	Operate successfully.

39.5.9 **status_t SDSPI_SwitchToHighSpeed (*sdspi_card_t * card*)**

This function can be called after SDSPI_Init function if target board's layout support >25MHZ spi baudrate, otherwise this function is useless. Be careful with call this function, code size and stack usage will be enlarge.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_Fail</i>	switch failed.
<i>kStatus_Success</i>	Operate successfully.

Chapter 40

CODEC Driver

40.1 Overview

The MCUXpresso SDK provides a codec abstraction driver interface to access codec register.

Modules

- [CODEC Common Driver](#)
- [CODEC I2C Driver](#)
- [CS42888 Driver](#)
- [DA7212 Driver](#)
- [SGTL5000 Driver](#)
- [WM8904 Driver](#)
- [WM8960 Driver](#)

40.2 CODEC Common Driver

40.2.1 Overview

The codec common driver provides a codec control abstraction interface.

Modules

- [CODEC Adapter](#)
- [CS42888 Adapter](#)
- [DA7212 Adapter](#)
- [SGTL5000 Adapter](#)
- [WM8904 Adapter](#)
- [WM8960 Adapter](#)

Data Structures

- struct [codec_config_t](#)
Initialize structure of the codec. [More...](#)
- struct [codec_capability_t](#)
codec capability [More...](#)
- struct [codec_handle_t](#)
Codec handle definition. [More...](#)

Macros

- #define [CODEC_VOLUME_MAX_VALUE](#) (100U)
codec maximum volume range

Enumerations

- enum {
 [kStatus_CODEC_NotSupport](#) = MAKE_STATUS(kStatusGroup_CODEC, 0U),
 [kStatus_CODEC_DeviceNotRegistered](#) = MAKE_STATUS(kStatusGroup_CODEC, 1U),
 [kStatus_CODEC_I2CBusInitialFailed](#),
 [kStatus_CODEC_I2CCommandTransferFailed](#) }

CODEC status.
- enum [codec_audio_protocol_t](#) {
 [kCODEC_BusI2S](#) = 0U,
 [kCODEC_BusLeftJustified](#) = 1U,
 [kCODEC_BusRightJustified](#) = 2U,
 [kCODEC_BusPCMA](#) = 3U,
 [kCODEC_BusPCMB](#) = 4U,
 [kCODEC_BusTDM](#) = 5U }

AUDIO format definition.

- enum {

kCODEC_AudioSampleRate8KHz = 8000U,
 kCODEC_AudioSampleRate11025Hz = 11025U,
 kCODEC_AudioSampleRate12KHz = 12000U,
 kCODEC_AudioSampleRate16KHz = 16000U,
 kCODEC_AudioSampleRate22050Hz = 22050U,
 kCODEC_AudioSampleRate24KHz = 24000U,
 kCODEC_AudioSampleRate32KHz = 32000U,
 kCODEC_AudioSampleRate44100Hz = 44100U,
 kCODEC_AudioSampleRate48KHz = 48000U,
 kCODEC_AudioSampleRate96KHz = 96000U,
 kCODEC_AudioSampleRate192KHz = 192000U,
 kCODEC_AudioSampleRate384KHz = 384000U }

audio sample rate definition

- enum {

kCODEC_AudioBitWidth16bit = 16U,
 kCODEC_AudioBitWidth20bit = 20U,
 kCODEC_AudioBitWidth24bit = 24U,
 kCODEC_AudioBitWidth32bit = 32U }

audio bit width

- enum codec_module_t {

kCODEC_ModuleADC = 0U,
 kCODEC_ModuleDAC = 1U,
 kCODEC_ModulePGA = 2U,
 kCODEC_ModuleHeadphone = 3U,
 kCODEC_ModuleSpeaker = 4U,
 kCODEC_ModuleLinein = 5U,
 kCODEC_ModuleLineout = 6U,
 kCODEC_ModuleVref = 7U,
 kCODEC_ModuleMicbias = 8U,
 kCODEC_ModuleMic = 9U,
 kCODEC_ModuleI2SIn = 10U,
 kCODEC_ModuleI2SOut = 11U,
 kCODEC_ModuleMixer = 12U }

audio codec module

- enum codec_module_ctrl_cmd_t { kCODEC_ModuleSwitchI2SInInterface = 0U }

audio codec module control cmd

- enum {

kCODEC_ModuleI2SInInterfacePCM = 0U,
 kCODEC_ModuleI2SInInterfaceDSD = 1U }

audio codec module digital interface

- enum {

- ```

kCODEC_RecordSourceDifferentialLine = 1U,
kCODEC_RecordSourceLineInput = 2U,
kCODEC_RecordSourceDifferentialMic = 4U,
kCODEC_RecordSourceDigitalMic = 8U,
kCODEC_RecordSourceSingleEndMic = 16U }
 audio codec module record source value
```
- enum {
 

```

kCODEC_RecordChannelLeft1 = 1U,
kCODEC_RecordChannelLeft2 = 2U,
kCODEC_RecordChannelLeft3 = 4U,
kCODEC_RecordChannelRight1 = 1U,
kCODEC_RecordChannelRight2 = 2U,
kCODEC_RecordChannelRight3 = 4U,
kCODEC_RecordChannelDifferentialPositive1 = 1U,
kCODEC_RecordChannelDifferentialPositive2 = 2U,
kCODEC_RecordChannelDifferentialPositive3 = 4U,
kCODEC_RecordChannelDifferentialNegative1 = 8U,
kCODEC_RecordChannelDifferentialNegative2 = 16U,
kCODEC_RecordChannelDifferentialNegative3 = 32U }
```

*audio codec record channel*
- enum {
 

```

kCODEC_PlaySourcePGA = 1U,
kCODEC_PlaySourceInput = 2U,
kCODEC_PlaySourceDAC = 4U,
kCODEC_PlaySourceMixerIn = 1U,
kCODEC_PlaySourceMixerInLeft = 2U,
kCODEC_PlaySourceMixerInRight = 4U,
kCODEC_PlaySourceAux = 8U }
```

*audio codec module play source value*
- enum {
 

```

kCODEC_PlayChannelHeadphoneLeft = 1U,
kCODEC_PlayChannelHeadphoneRight = 2U,
kCODEC_PlayChannelSpeakerLeft = 4U,
kCODEC_PlayChannelSpeakerRight = 8U,
kCODEC_PlayChannelLineOutLeft = 16U,
kCODEC_PlayChannelLineOutRight = 32U,
kCODEC_PlayChannelLeft0 = 1U,
kCODEC_PlayChannelRight0 = 2U,
kCODEC_PlayChannelLeft1 = 4U,
kCODEC_PlayChannelRight1 = 8U,
kCODEC_PlayChannelLeft2 = 16U,
kCODEC_PlayChannelRight2 = 32U,
kCODEC_PlayChannelLeft3 = 64U,
kCODEC_PlayChannelRight3 = 128U }
```

*codec play channel*
- enum {

```
kCODEC_VolumeHeadphoneLeft = 1U,
kCODEC_VolumeHeadphoneRight = 2U,
kCODEC_VolumeSpeakerLeft = 4U,
kCODEC_VolumeSpeakerRight = 8U,
kCODEC_VolumeLineOutLeft = 16U,
kCODEC_VolumeLineOutRight = 32U,
kCODEC_VolumeLeft0 = 1UL << 0U,
kCODEC_VolumeRight0 = 1UL << 1U,
kCODEC_VolumeLeft1 = 1UL << 2U,
kCODEC_VolumeRight1 = 1UL << 3U,
kCODEC_VolumeLeft2 = 1UL << 4U,
kCODEC_VolumeRight2 = 1UL << 5U,
kCODEC_VolumeLeft3 = 1UL << 6U,
kCODEC_VolumeRight3 = 1UL << 7U,
kCODEC_VolumeDAC = 1UL << 8U }
```

*codec volume setting*

- enum {

```

kCODEC_SupportModuleADC = 1U << 0U,
kCODEC_SupportModuleDAC = 1U << 1U,
kCODEC_SupportModulePGA = 1U << 2U,
kCODEC_SupportModuleHeadphone = 1U << 3U,
kCODEC_SupportModuleSpeaker = 1U << 4U,
kCODEC_SupportModuleLinein = 1U << 5U,
kCODEC_SupportModuleLineout = 1U << 6U,
kCODEC_SupportModuleVref = 1U << 7U,
kCODEC_SupportModuleMicbias = 1U << 8U,
kCODEC_SupportModuleMic = 1U << 9U,
kCODEC_SupportModuleI2SIn = 1U << 10U,
kCODEC_SupportModuleI2SOut = 1U << 11U,
kCODEC_SupportModuleMixer = 1U << 12U,
kCODEC_SupportModuleI2SInSwitchInterface = 1U << 13U,
kCODEC_SupportPlayChannelLeft0 = 1U << 0U,
kCODEC_SupportPlayChannelRight0 = 1U << 1U,
kCODEC_SupportPlayChannelLeft1 = 1U << 2U,
kCODEC_SupportPlayChannelRight1 = 1U << 3U,
kCODEC_SupportPlayChannelLeft2 = 1U << 4U,
kCODEC_SupportPlayChannelRight2 = 1U << 5U,
kCODEC_SupportPlayChannelLeft3 = 1U << 6U,
kCODEC_SupportPlayChannelRight3 = 1U << 7U,
kCODEC_SupportPlaySourcePGA = 1U << 8U,
kCODEC_SupportPlaySourceInput = 1U << 9U,
kCODEC_SupportPlaySourceDAC = 1U << 10U,
kCODEC_SupportPlaySourceMixerIn = 1U << 11U,
kCODEC_SupportPlaySourceMixerInLeft = 1U << 12U,
kCODEC_SupportPlaySourceMixerInRight = 1U << 13U,
kCODEC_SupportPlaySourceAux = 1U << 14U,
kCODEC_SupportRecordSourceDifferentialLine = 1U << 0U,
kCODEC_SupportRecordSourceLineInput = 1U << 1U,
kCODEC_SupportRecordSourceDifferentialMic = 1U << 2U,
kCODEC_SupportRecordSourceDigitalMic = 1U << 3U,
kCODEC_SupportRecordSourceSingleEndMic = 1U << 4U,
kCODEC_SupportRecordChannelLeft1 = 1U << 6U,
kCODEC_SupportRecordChannelLeft2 = 1U << 7U,
kCODEC_SupportRecordChannelLeft3 = 1U << 8U,
kCODEC_SupportRecordChannelRight1 = 1U << 9U,
kCODEC_SupportRecordChannelRight2 = 1U << 10U,
kCODEC_SupportRecordChannelRight3 = 1U << 11U }

```

*audio codec capability*

## Functions

- `status_t CODEC_Init (codec_handle_t *handle, codec_config_t *config)`  
*Codec initialization.*
- `status_t CODEC_Deinit (codec_handle_t *handle)`  
*Codec de-initilization.*
- `status_t CODEC_SetFormat (codec_handle_t *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `status_t CODEC_ModuleControl (codec_handle_t *handle, codec_module_ctrl_cmd_t cmd, uint32_t data)`  
*codec module control.*
- `status_t CODEC_SetVolume (codec_handle_t *handle, uint32_t channel, uint32_t volume)`  
*set audio codec pl volume.*
- `status_t CODEC_SetMute (codec_handle_t *handle, uint32_t channel, bool mute)`  
*set audio codec module mute.*
- `status_t CODEC_SetPower (codec_handle_t *handle, codec_module_t module, bool powerOn)`  
*set audio codec power.*
- `status_t CODEC_SetRecord (codec_handle_t *handle, uint32_t recordSource)`  
*codec set record source.*
- `status_t CODEC_SetRecordChannel (codec_handle_t *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)`  
*codec set record channel.*
- `status_t CODEC_SetPlay (codec_handle_t *handle, uint32_t playSource)`  
*codec set play source.*

## Driver version

- `#define FSL_CODEC_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))`  
*CLOCK driver version 2.3.1.*

### 40.2.2 Data Structure Documentation

#### 40.2.2.1 struct codec\_config\_t

##### Data Fields

- `uint32_t codecDevType`  
*codec type*
- `void *codecDevConfig`  
*Codec device specific configuration.*

### 40.2.2.2 struct codec\_capability\_t

#### Data Fields

- `uint32_t codecModuleCapability`  
*codec module capability*
- `uint32_t codecPlayCapability`  
*codec play capability*
- `uint32_t codecRecordCapability`  
*codec record capability*
- `uint32_t codecVolumeCapability`  
*codec volume capability*

### 40.2.2.3 struct \_codec\_handle

codec handle declaration

- Application should allocate a buffer with CODEC\_HANDLE\_SIZE for handle definition, such as `uint8_t codecHandleBuffer[CODEC_HANDLE_SIZE]; codec_handle_t *codecHandle = codecHandleBuffer;`

#### Data Fields

- `codec_config_t * codecConfig`  
*codec configuration function pointer*
- `const codec_capability_t * codecCapability`  
*codec capability*
- `uint8_t codecDevHandle [HAL_CODEC_HANDLER_SIZE]`  
*codec device handle*

### 40.2.3 Macro Definition Documentation

#### 40.2.3.1 #define FSL\_CODEC\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 1))

### 40.2.4 Enumeration Type Documentation

#### 40.2.4.1 anonymous enum

Enumerator

`kStatus_CODEC_NotSupport` CODEC not support status.

`kStatus_CODEC_DeviceNotRegistered` CODEC device register failed status.

`kStatus_CODEC_I2CBusInitialFailed` CODEC i2c bus initialization failed status.

`kStatus_CODEC_I2CCommandTransferFailed` CODEC i2c bus command transfer failed status.

#### 40.2.4.2 enum codec\_audio\_protocol\_t

Enumerator

- kCODEC\_BusI2S* I2S type.
- kCODEC\_BusLeftJustified* Left justified mode.
- kCODEC\_BusRightJustified* Right justified mode.
- kCODEC\_BusPCMA* DSP/PCM A mode.
- kCODEC\_BusPCMB* DSP/PCM B mode.
- kCODEC\_BusTDM* TDM mode.

#### 40.2.4.3 anonymous enum

Enumerator

- kCODEC\_AudioSampleRate8KHz* Sample rate 8000 Hz.
- kCODEC\_AudioSampleRate11025Hz* Sample rate 11025 Hz.
- kCODEC\_AudioSampleRate12KHz* Sample rate 12000 Hz.
- kCODEC\_AudioSampleRate16KHz* Sample rate 16000 Hz.
- kCODEC\_AudioSampleRate22050Hz* Sample rate 22050 Hz.
- kCODEC\_AudioSampleRate24KHz* Sample rate 24000 Hz.
- kCODEC\_AudioSampleRate32KHz* Sample rate 32000 Hz.
- kCODEC\_AudioSampleRate44100Hz* Sample rate 44100 Hz.
- kCODEC\_AudioSampleRate48KHz* Sample rate 48000 Hz.
- kCODEC\_AudioSampleRate96KHz* Sample rate 96000 Hz.
- kCODEC\_AudioSampleRate192KHz* Sample rate 192000 Hz.
- kCODEC\_AudioSampleRate384KHz* Sample rate 384000 Hz.

#### 40.2.4.4 anonymous enum

Enumerator

- kCODEC\_AudioBitWidth16bit* audio bit width 16
- kCODEC\_AudioBitWidth20bit* audio bit width 20
- kCODEC\_AudioBitWidth24bit* audio bit width 24
- kCODEC\_AudioBitWidth32bit* audio bit width 32

#### 40.2.4.5 enum codec\_module\_t

Enumerator

- kCODEC\_ModuleADC* codec module ADC
- kCODEC\_ModuleDAC* codec module DAC
- kCODEC\_ModulePGA* codec module PGA
- kCODEC\_ModuleHeadphone* codec module headphone

*kCODEC\_ModuleSpeaker* codec module speaker  
*kCODEC\_ModuleLinein* codec module linein  
*kCODEC\_ModuleLineout* codec module lineout  
*kCODEC\_ModuleVref* codec module VREF  
*kCODEC\_ModuleMicbias* codec module MIC BIAS  
*kCODEC\_ModuleMic* codec module MIC  
*kCODEC\_ModuleI2SIn* codec module I2S in  
*kCODEC\_ModuleI2SOut* codec module I2S out  
*kCODEC\_ModuleMixer* codec module mixer

#### 40.2.4.6 enum codec\_module\_ctrl\_cmd\_t

Enumerator

*kCODEC\_ModuleSwitchI2SInInterface* module digital interface siwtch.

#### 40.2.4.7 anonymous enum

Enumerator

*kCODEC\_ModuleI2SInInterfacePCM* Pcm interface.  
*kCODEC\_ModuleI2SInInterfaceDSD* DSD interface.

#### 40.2.4.8 anonymous enum

Enumerator

*kCODEC\_RecordSourceDifferentialLine* record source from differential line  
*kCODEC\_RecordSourceLineInput* record source from line input  
*kCODEC\_RecordSourceDifferentialMic* record source from differential mic  
*kCODEC\_RecordSourceDigitalMic* record source from digital microphone  
*kCODEC\_RecordSourceSingleEndMic* record source from single microphone

#### 40.2.4.9 anonymous enum

Enumerator

*kCODEC\_RecordChannelLeft1* left record channel 1  
*kCODEC\_RecordChannelLeft2* left record channel 2  
*kCODEC\_RecordChannelLeft3* left record channel 3  
*kCODEC\_RecordChannelRight1* right record channel 1  
*kCODEC\_RecordChannelRight2* right record channel 2  
*kCODEC\_RecordChannelRight3* right record channel 3  
*kCODEC\_RecordChannelDifferentialPositive1* differential positive record channel 1

*kCODEC\_RecordChannelDifferentialPositive2* differential positive record channel 2  
*kCODEC\_RecordChannelDifferentialPositive3* differential positive record channel 3  
*kCODEC\_RecordChannelDifferentialNegative1* differential negative record channel 1  
*kCODEC\_RecordChannelDifferentialNegative2* differential negative record channel 2  
*kCODEC\_RecordChannelDifferentialNegative3* differential negative record channel 3

#### 40.2.4.10 anonymous enum

Enumerator

*kCODEC\_PlaySourcePGA* play source PGA, bypass ADC  
*kCODEC\_PlaySourceInput* play source Input3  
*kCODEC\_PlaySourceDAC* play source DAC  
*kCODEC\_PlaySourceMixerIn* play source mixer in  
*kCODEC\_PlaySourceMixerInLeft* play source mixer in left  
*kCODEC\_PlaySourceMixerInRight* play source mixer in right  
*kCODEC\_PlaySourceAux* play source mixer in AUX

#### 40.2.4.11 anonymous enum

Enumerator

*kCODEC\_PlayChannelHeadphoneLeft* play channel headphone left  
*kCODEC\_PlayChannelHeadphoneRight* play channel headphone right  
*kCODEC\_PlayChannelSpeakerLeft* play channel speaker left  
*kCODEC\_PlayChannelSpeakerRight* play channel speaker right  
*kCODEC\_PlayChannelLineOutLeft* play channel lineout left  
*kCODEC\_PlayChannelLineOutRight* play channel lineout right  
*kCODEC\_PlayChannelLeft0* play channel left0  
*kCODEC\_PlayChannelRight0* play channel right0  
*kCODEC\_PlayChannelLeft1* play channel left1  
*kCODEC\_PlayChannelRight1* play channel right1  
*kCODEC\_PlayChannelLeft2* play channel left2  
*kCODEC\_PlayChannelRight2* play channel right2  
*kCODEC\_PlayChannelLeft3* play channel left3  
*kCODEC\_PlayChannelRight3* play channel right3

#### 40.2.4.12 anonymous enum

Enumerator

*kCODEC\_VolumeHeadphoneLeft* headphone left volume  
*kCODEC\_VolumeHeadphoneRight* headphone right volume  
*kCODEC\_VolumeSpeakerLeft* speaker left volume  
*kCODEC\_VolumeSpeakerRight* speaker right volume

*kCODEC\_VolumeLineOutLeft* lineout left volume  
*kCODEC\_VolumeLineOutRight* lineout right volume  
*kCODEC\_VolumeLeft0* left0 volume  
*kCODEC\_VolumeRight0* right0 volume  
*kCODEC\_VolumeLeft1* left1 volume  
*kCODEC\_VolumeRight1* right1 volume  
*kCODEC\_VolumeLeft2* left2 volume  
*kCODEC\_VolumeRight2* right2 volume  
*kCODEC\_VolumeLeft3* left3 volume  
*kCODEC\_VolumeRight3* right3 volume  
*kCODEC\_VolumeDAC* dac volume

#### 40.2.4.13 anonymous enum

Enumerator

*kCODEC\_SupportModuleADC* codec capability of module ADC  
*kCODEC\_SupportModuleDAC* codec capability of module DAC  
*kCODEC\_SupportModulePGA* codec capability of module PGA  
*kCODEC\_SupportModuleHeadphone* codec capability of module headphone  
*kCODEC\_SupportModuleSpeaker* codec capability of module speaker  
*kCODEC\_SupportModuleLinein* codec capability of module linein  
*kCODEC\_SupportModuleLineout* codec capability of module lineout  
*kCODEC\_SupportModuleVref* codec capability of module vref  
*kCODEC\_SupportModuleMicbias* codec capability of module mic bias  
*kCODEC\_SupportModuleMic* codec capability of module mic bias  
*kCODEC\_SupportModuleI2SIn* codec capability of module I2S in  
*kCODEC\_SupportModuleI2SOut* codec capability of module I2S out  
*kCODEC\_SupportModuleMixer* codec capability of module mixer  
*kCODEC\_SupportModuleI2SInSwitchInterface* codec capability of module I2S in switch interface  
  
*kCODEC\_SupportPlayChannelLeft0* codec capability of play channel left 0  
*kCODEC\_SupportPlayChannelRight0* codec capability of play channel right 0  
*kCODEC\_SupportPlayChannelLeft1* codec capability of play channel left 1  
*kCODEC\_SupportPlayChannelRight1* codec capability of play channel right 1  
*kCODEC\_SupportPlayChannelLeft2* codec capability of play channel left 2  
*kCODEC\_SupportPlayChannelRight2* codec capability of play channel right 2  
*kCODEC\_SupportPlayChannelLeft3* codec capability of play channel left 3  
*kCODEC\_SupportPlayChannelRight3* codec capability of play channel right 3  
*kCODEC\_SupportPlaySourcePGA* codec capability of set playback source PGA  
*kCODEC\_SupportPlaySourceInput* codec capability of set playback source INPUT  
*kCODEC\_SupportPlaySourceDAC* codec capability of set playback source DAC  
*kCODEC\_SupportPlaySourceMixerIn* codec capability of set play source Mixer in  
*kCODEC\_SupportPlaySourceMixerInLeft* codec capability of set play source Mixer in left  
*kCODEC\_SupportPlaySourceMixerInRight* codec capability of set play source Mixer in right

***kCODEC\_SupportPlaySourceAux*** codec capability of set play source aux

***kCODEC\_SupportRecordSourceDifferentialLine*** codec capability of record source differential line

***kCODEC\_SupportRecordSourceLineInput*** codec capability of record source line input

***kCODEC\_SupportRecordSourceDifferentialMic*** codec capability of record source differential mic

***kCODEC\_SupportRecordSourceDigitalMic*** codec capability of record digital mic

***kCODEC\_SupportRecordSourceSingleEndMic*** codec capability of single end mic

***kCODEC\_SupportRecordChannelLeft1*** left record channel 1

***kCODEC\_SupportRecordChannelLeft2*** left record channel 2

***kCODEC\_SupportRecordChannelLeft3*** left record channel 3

***kCODEC\_SupportRecordChannelRight1*** right record channel 1

***kCODEC\_SupportRecordChannelRight2*** right record channel 2

***kCODEC\_SupportRecordChannelRight3*** right record channel 3

## 40.2.5 Function Documentation

### 40.2.5.1 status\_t CODEC\_Init ( ***codec\_handle\_t \* handle***, ***codec\_config\_t \* config*** )

Parameters

|               |                       |
|---------------|-----------------------|
| <i>handle</i> | codec handle.         |
| <i>config</i> | codec configurations. |

Returns

kStatus\_Success is success, else de-initial failed.

### 40.2.5.2 status\_t CODEC\_Deinit ( ***codec\_handle\_t \* handle*** )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

### 40.2.5.3 status\_t CODEC\_SetFormat ( ***codec\_handle\_t \* handle***, ***uint32\_t mclk***, ***uint32\_t sampleRate***, ***uint32\_t bitWidth*** )

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

Returns

kStatus\_Success is success, else configure failed.

#### 40.2.5.4 status\_t CODEC\_ModuleControl ( *codec\_handle\_t \* handle*, *codec\_module\_ctrl\_cmd\_t cmd*, *uint32\_t data* )

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature.

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.2.5.5 status\_t CODEC\_SetVolume ( *codec\_handle\_t \* handle*, *uint32\_t channel*, *uint32\_t volume* )

Parameters

|                |                                                                                                                  |
|----------------|------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>  | codec handle.                                                                                                    |
| <i>channel</i> | audio codec volume channel, can be a value or combine value of _codec_volume_-capability or _codec_play_channel. |
| <i>volume</i>  | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.                                       |

Returns

kStatus\_Success is success, else configure failed.

#### 40.2.5.6 status\_t CODEC\_SetMute ( *codec\_handle\_t \* handle*, *uint32\_t channel*, *bool mute* )

Parameters

|                |                                                                                                                  |
|----------------|------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>  | codec handle.                                                                                                    |
| <i>channel</i> | audio codec volume channel, can be a value or combine value of _codec_volume_-capability or _codec_play_channel. |
| <i>mute</i>    | true is mute, false is unmute.                                                                                   |

Returns

kStatus\_Success is success, else configure failed.

#### 40.2.5.7 status\_t CODEC\_SetPower ( *codec\_handle\_t \* handle*, *codec\_module\_t module*, *bool powerOn* )

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.2.5.8 status\_t CODEC\_SetRecord ( *codec\_handle\_t \* handle*, *uint32\_t recordSource* )

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.2.5.9 status\_t CODEC\_SetRecordChannel ( *codec\_handle\_t \* handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel* )

Parameters

|                            |                                                                                                                         |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                           |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.2.5.10 status\_t CODEC\_SetPlay ( *codec\_handle\_t \* handle, uint32\_t playSource* )

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

## 40.3 CODEC I2C Driver

### 40.3.1 Overview

The codec common driver provides a codec control abstraction interface.

## Data Structures

- struct `codec_i2c_config_t`  
*CODEC I2C configurations structure. [More...](#)*

## Macros

- #define `CODEC_I2C_MASTER_HANDLER_SIZE` HAL\_I2C\_MASTER\_HANDLE\_SIZE  
*codec i2c handler*

## Enumerations

- enum `codec_reg_addr_t` {
   
`kCODEC_RegAddr8Bit` = 1U,  
`kCODEC_RegAddr16Bit` = 2U }  
*CODEC device register address type.*
- enum `codec_reg_width_t` {
   
`kCODEC_RegWidth8Bit` = 1U,  
`kCODEC_RegWidth16Bit` = 2U,  
`kCODEC_RegWidth32Bit` = 4U }  
*CODEC device register width.*

## Functions

- `status_t CODEC_I2C_Init` (void \*handle, uint32\_t i2cInstance, uint32\_t i2cBaudrate, uint32\_t i2cSourceClockHz)  
*Codec i2c bus initialization.*
- `status_t CODEC_I2C_Deinit` (void \*handle)  
*Codec i2c de-initilization.*
- `status_t CODEC_I2C_Send` (void \*handle, uint8\_t deviceAddress, uint32\_t subAddress, uint8\_t subaddressSize, uint8\_t \*txBuff, uint8\_t txBuffSize)  
*codec i2c send function.*
- `status_t CODEC_I2C_Receive` (void \*handle, uint8\_t deviceAddress, uint32\_t subAddress, uint8\_t subaddressSize, uint8\_t \*rxBuff, uint8\_t rxBuffSize)  
*codec i2c receive function.*

## 40.3.2 Data Structure Documentation

### 40.3.2.1 struct codec\_i2c\_config\_t

#### Data Fields

- `uint32_t codecI2CInstance`  
*i2c bus instance*
- `uint32_t codecI2CSourceClock`  
*i2c bus source clock frequency*

## 40.3.3 Enumeration Type Documentation

### 40.3.3.1 enum codec\_reg\_addr\_t

Enumerator

`kCODEC_RegAddr8Bit` 8-bit register address.  
`kCODEC_RegAddr16Bit` 16-bit register address.

### 40.3.3.2 enum codec\_reg\_width\_t

Enumerator

`kCODEC_RegWidth8Bit` 8-bit register width.  
`kCODEC_RegWidth16Bit` 16-bit register width.  
`kCODEC_RegWidth32Bit` 32-bit register width.

## 40.3.4 Function Documentation

### 40.3.4.1 status\_t CODEC\_I2C\_Init ( void \* handle, uint32\_t i2cInstance, uint32\_t i2cBaudrate, uint32\_t i2cSourceClockHz )

Parameters

|                          |                                                                     |
|--------------------------|---------------------------------------------------------------------|
| <code>handle</code>      | i2c master handle.                                                  |
| <code>i2cInstance</code> | instance number of the i2c bus, such as 0 is corresponding to I2C0. |

|                          |                             |
|--------------------------|-----------------------------|
| <i>i2cBaudrate</i>       | i2c baudrate.               |
| <i>i2cSource-ClockHz</i> | i2c source clock frequency. |

Returns

kStatus\_HAL\_I2cSuccess is success, else initial failed.

#### 40.3.4.2 status\_t CODEC\_I2C\_Deinit ( void \* *handle* )

Parameters

|               |                    |
|---------------|--------------------|
| <i>handle</i> | i2c master handle. |
|---------------|--------------------|

Returns

kStatus\_HAL\_I2cSuccess is success, else deinitial failed.

#### 40.3.4.3 status\_t CODEC\_I2C\_Send ( void \* *handle*, uint8\_t *deviceAddress*, uint32\_t *subAddress*, uint8\_t *subaddressSize*, uint8\_t \* *txBuff*, uint8\_t *txBuffSize* )

Parameters

|                       |                         |
|-----------------------|-------------------------|
| <i>handle</i>         | i2c master handle.      |
| <i>deviceAddress</i>  | codec device address.   |
| <i>subAddress</i>     | register address.       |
| <i>subaddressSize</i> | register address width. |
| <i>txBuff</i>         | tx buffer pointer.      |
| <i>txBuffSize</i>     | tx buffer size.         |

Returns

kStatus\_HAL\_I2cSuccess is success, else send failed.

#### 40.3.4.4 status\_t CODEC\_I2C\_Receive ( void \* *handle*, uint8\_t *deviceAddress*, uint32\_t *subAddress*, uint8\_t *subaddressSize*, uint8\_t \* *rxBuff*, uint8\_t *rxBuffSize* )

## Parameters

|                       |                         |
|-----------------------|-------------------------|
| <i>handle</i>         | i2c master handle.      |
| <i>deviceAddress</i>  | codec device address.   |
| <i>subAddress</i>     | register address.       |
| <i>subaddressSize</i> | register address width. |
| <i>rxBuff</i>         | rx buffer pointer.      |
| <i>rxBuffSize</i>     | rx buffer size.         |

## Returns

kStatus\_HAL\_I2cSuccess is success, else receive failed.

## 40.4 CS42888 Driver

### 40.4.1 Overview

The cs42888 driver provides a codec control interface.

### Data Structures

- struct `cs42888_audio_format_t`  
*cs42888 audio format* [More...](#)
- struct `cs42888_config_t`  
*Initialize structure of CS42888.* [More...](#)
- struct `cs42888_handle_t`  
*cs42888 handler* [More...](#)

### Macros

- #define `CS42888_I2C_HANDLER_SIZE` CODEC\_I2C\_MASTER\_HANDLER\_SIZE  
*CS42888 handle size.*
- #define `CS42888_ID` 0x01U  
*Define the register address of CS42888.*
- #define `CS42888_AOUT_MAX_VOLUME_VALUE` 0xFFU  
*CS42888 volume setting range.*
- #define `CS42888_CACHEREGNUM` 28U  
*Cache register number.*
- #define `CS42888_I2C_ADDR` 0x48U  
*CS42888 I2C address.*
- #define `CS42888_I2C_BITRATE` (100000U)  
*CS42888 I2C baudrate.*

### Typedefs

- typedef void(\* `cs42888_reset` )(bool state)  
*cs42888 reset function pointer*

### Enumerations

- enum `cs42888_func_mode` {
   
`kCS42888_ModeMasterSSM` = 0x0,  
`kCS42888_ModeMasterDSM` = 0x1,  
`kCS42888_ModeMasterQSM` = 0x2,  
`kCS42888_ModeSlave` = 0x3
 }
- CS42888 support modes.*

- enum `cs42888_module_t` {
   
kCS42888\_ModuleDACPair1 = 0x2,  
 kCS42888\_ModuleDACPair2 = 0x4,  
 kCS42888\_ModuleDACPair3 = 0x8,  
 kCS42888\_ModuleDACPair4 = 0x10,  
 kCS42888\_ModuleADCPair1 = 0x20,  
 kCS42888\_ModuleADCPair2 = 0x40 }

*Modules in CS42888 board.*

- enum `cs42888_bus_t` {
   
kCS42888\_BusLeftJustified = 0x0,  
 kCS42888\_BusI2S = 0x1,  
 kCS42888\_BusRightJustified = 0x2,  
 kCS42888\_BusOL1 = 0x4,  
 kCS42888\_BusOL2 = 0x5,  
 kCS42888\_BusTDM = 0x6 }

*CS42888 supported audio bus type.*

- enum {
   
kCS42888\_AOUT1 = 1U,  
 kCS42888\_AOUT2 = 2U,  
 kCS42888\_AOUT3 = 3U,  
 kCS42888\_AOUT4 = 4U,  
 kCS42888\_AOUT5 = 5U,  
 kCS42888\_AOUT6 = 6U,  
 kCS42888\_AOUT7 = 7U,  
 kCS42888\_AOUT8 = 8U }

*CS42888 play channel.*

## Functions

- `status_t CS42888_Init (cs42888_handle_t *handle, cs42888_config_t *config)`  
*CS42888 initialize function.*
- `status_t CS42888_Deinit (cs42888_handle_t *handle)`  
*Deinit the CS42888 codec.*
- `status_t CS42888_SetProtocol (cs42888_handle_t *handle, cs42888_bus_t protocol, uint32_t bitWidth)`  
*Set the audio transfer protocol.*
- `void CS42888_SetFuncMode (cs42888_handle_t *handle, cs42888_func_mode mode)`  
*Set CS42888 to differernt working mode.*
- `status_t CS42888_SelectFunctionalMode (cs42888_handle_t *handle, cs42888_func_mode adcMode, cs42888_func_mode dacMode)`  
*Set CS42888 to differernt functional mode.*
- `status_t CS42888_SetAOUTVolume (cs42888_handle_t *handle, uint8_t channel, uint8_t volume)`  
*Set the volume of different modules in CS42888.*
- `status_t CS42888_SetAINVolume (cs42888_handle_t *handle, uint8_t channel, uint8_t volume)`  
*Set the volume of different modules in CS42888.*
- `uint8_t CS42888_GetAOUTVolume (cs42888_handle_t *handle, uint8_t channel)`

- `uint8_t CS42888_GetAINVolume (cs42888_handle_t *handle, uint8_t channel)`  
*Get the volume of different AIN channel in CS42888.*
- `status_t CS42888_SetMute (cs42888_handle_t *handle, uint8_t channelMask)`  
*Mute modules in CS42888.*
- `status_t CS42888_SetChannelMute (cs42888_handle_t *handle, uint8_t channel, bool isMute)`  
*Mute channel modules in CS42888.*
- `status_t CS42888_SetModule (cs42888_handle_t *handle, cs42888_module_t module, bool isEnabled)`  
*Enable/disable expected devices.*
- `status_t CS42888_ConfigDataFormat (cs42888_handle_t *handle, uint32_t mclk, uint32_t sample_rate, uint32_t bits)`  
*Configure the data format of audio data.*
- `status_t CS42888_WriteReg (cs42888_handle_t *handle, uint8_t reg, uint8_t val)`  
*Write register to CS42888 using I2C.*
- `status_t CS42888_ReadReg (cs42888_handle_t *handle, uint8_t reg, uint8_t *val)`  
*Read register from CS42888 using I2C.*
- `status_t CS42888_ModifyReg (cs42888_handle_t *handle, uint8_t reg, uint8_t mask, uint8_t val)`  
*Modify some bits in the register using I2C.*

## Driver version

- `#define FSL_CS42888_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))`  
*cs42888 driver version 2.1.3.*

### 40.4.2 Data Structure Documentation

#### 40.4.2.1 struct cs42888\_audio\_format\_t

##### Data Fields

- `uint32_t mclk_HZ`  
*master clock frequency*
- `uint32_t sampleRate`  
*sample rate*
- `uint32_t bitWidth`  
*bit width*

#### 40.4.2.2 struct cs42888\_config\_t

##### Data Fields

- `cs42888_bus_t bus`  
*Audio transfer protocol.*
- `cs42888_audio_format_t format`  
*cs42888 audio format*

- `cs42888_func_mode ADCMode`  
*CS42888 ADC function mode.*
- `cs42888_func_mode DACMode`  
*CS42888 DAC function mode.*
- `bool master`  
*true is master, false is slave*
- `codec_i2c_config_t i2cConfig`  
*i2c bus configuration*
- `uint8_t slaveAddress`  
*slave address*
- `cs42888_reset reset`  
*reset function pointer*

## Field Documentation

- (1) `cs42888_func_mode cs42888_config_t::ADCMode`
- (2) `cs42888_func_mode cs42888_config_t::DACMode`

### 40.4.2.3 struct cs42888\_handle\_t

#### Data Fields

- `cs42888_config_t * config`  
*cs42888 config pointer*
- `uint8_t i2cHandle [CS42888_I2C_HANDLER_SIZE]`  
*i2c handle pointer*

### 40.4.3 Macro Definition Documentation

#### 40.4.3.1 #define FSL\_CS42888\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 3))

#### 40.4.3.2 #define CS42888\_ID 0x01U

#### 40.4.3.3 #define CS42888\_I2C\_ADDR 0x48U

### 40.4.4 Enumeration Type Documentation

#### 40.4.4.1 enum cs42888\_func\_mode

Enumerator

- kCS42888\_ModeMasterSSM* master single speed mode
- kCS42888\_ModeMasterDSM* master dual speed mode
- kCS42888\_ModeMasterQSM* master quad speed mode
- kCS42888\_ModeSlave* master single speed mode

#### 40.4.4.2 enum cs42888\_module\_t

Enumerator

- kCS42888\_ModuleDACPair1* DAC pair1 (AOUT1 and AOUT2) module in CS42888.
- kCS42888\_ModuleDACPair2* DAC pair2 (AOUT3 and AOUT4) module in CS42888.
- kCS42888\_ModuleDACPair3* DAC pair3 (AOUT5 and AOUT6) module in CS42888.
- kCS42888\_ModuleDACPair4* DAC pair4 (AOUT7 and AOUT8) module in CS42888.
- kCS42888\_ModuleADCPair1* ADC pair1 (AIN1 and AIN2) module in CS42888.
- kCS42888\_ModuleADCPair2* ADC pair2 (AIN3 and AIN4) module in CS42888.

#### 40.4.4.3 enum cs42888\_bus\_t

Enumerator

- kCS42888\_BusLeftJustified* Left justified format, up to 24 bits.
- kCS42888\_BusI2S* I2S format, up to 24 bits.
- kCS42888\_BusRightJustified* Right justified, can support 16bits and 24 bits.
- kCS42888\_BusOL1* One-Line #1 mode.
- kCS42888\_BusOL2* One-Line #2 mode.
- kCS42888\_BusTDM* TDM mode.

#### 40.4.4.4 anonymous enum

Enumerator

- kCS42888\_AOUT1* aout1
- kCS42888\_AOUT2* aout2
- kCS42888\_AOUT3* aout3
- kCS42888\_AOUT4* aout4
- kCS42888\_AOUT5* aout5
- kCS42888\_AOUT6* aout6
- kCS42888\_AOUT7* aout7
- kCS42888\_AOUT8* aout8

### 40.4.5 Function Documentation

#### 40.4.5.1 status\_t CS42888\_Init ( *cs42888\_handle\_t \* handle*, *cs42888\_config\_t \* config* )

The second parameter is NULL to CS42888 in this version. If users want to change the settings, they have to use *cs42888\_write\_reg()* or *cs42888\_modify\_reg()* to set the register value of CS42888. Note: If the *codec\_config* is NULL, it would initialize CS42888 using default settings. The default setting: *codec\_config->bus* = *kCS42888\_BusI2S* *codec\_config->ADCmode* = *kCS42888\_ModeSlave* *codec\_config->DACmode* = *kCS42888\_ModeSlave*

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>handle</i> | CS42888 handle structure.        |
| <i>config</i> | CS42888 configuration structure. |

#### 40.4.5.2 status\_t CS42888\_Deinit ( cs42888\_handle\_t \* *handle* )

This function close all modules in CS42888 to save power.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>handle</i> | CS42888 handle structure pointer. |
|---------------|-----------------------------------|

#### 40.4.5.3 status\_t CS42888\_SetProtocol ( cs42888\_handle\_t \* *handle*, cs42888\_bus\_t *protocol*, uint32\_t *bitWidth* )

CS42888 only supports I2S, left justified, right justified, PCM A, PCM B format.

Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>handle</i>   | CS42888 handle structure.     |
| <i>protocol</i> | Audio data transfer protocol. |
| <i>bitWidth</i> | bit width                     |

#### 40.4.5.4 void CS42888\_SetFuncMode ( cs42888\_handle\_t \* *handle*, cs42888\_func\_mode *mode* )

**Deprecated** api, Do not use it anymore. It has been superceded by [CS42888\\_SelectFunctionalMode](#).

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>handle</i> | CS42888 handle structure.           |
| <i>mode</i>   | differenht working mode of CS42888. |

#### 40.4.5.5 status\_t CS42888\_SelectFunctionalMode ( cs42888\_handle\_t \* *handle*, cs42888\_func\_mode *adcMode*, cs42888\_func\_mode *dacMode* )

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>handle</i>  | CS42888 handle structure.           |
| <i>adcMode</i> | differenht working mode of CS42888. |
| <i>dacMode</i> | differenht working mode of CS42888. |

#### 40.4.5.6 status\_t CS42888\_SetAOUTVolume ( *cs42888\_handle\_t \* handle, uint8\_t channel, uint8\_t volume* )

This function would set the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

|                |                                |
|----------------|--------------------------------|
| <i>handle</i>  | CS42888 handle structure.      |
| <i>channel</i> | AOUT channel, it shall be 1~8. |
| <i>volume</i>  | Volume value need to be set.   |

#### 40.4.5.7 status\_t CS42888\_SetAINVolume ( *cs42888\_handle\_t \* handle, uint8\_t channel, uint8\_t volume* )

This function would set the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>handle</i>  | CS42888 handle structure.     |
| <i>channel</i> | AIN channel, it shall be 1~4. |
| <i>volume</i>  | Volume value need to be set.  |

#### 40.4.5.8 uint8\_t CS42888\_GetAOUTVolume ( *cs42888\_handle\_t \* handle, uint8\_t channel* )

This function gets the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

|                |                                |
|----------------|--------------------------------|
| <i>handle</i>  | CS42888 handle structure.      |
| <i>channel</i> | AOUT channel, it shall be 1~8. |

#### 40.4.5.9 `uint8_t CS42888_GetAINVolume ( cs42888_handle_t * handle, uint8_t channel )`

This function gets the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>handle</i>  | CS42888 handle structure.     |
| <i>channel</i> | AIN channel, it shall be 1~4. |

#### 40.4.5.10 `status_t CS42888_SetMute ( cs42888_handle_t * handle, uint8_t channelMask )`

Parameters

|                    |                                                                                                                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>      | CS42888 handle structure.                                                                                                                                                                              |
| <i>channelMask</i> | Channel mask for mute. Mute channel 0, it shall be 0x1, while mute channel 0 and 1, it shall be 0x3. Mute all channel, it shall be 0xFF. Each bit represent one channel, 1 means mute, 0 means unmute. |

#### 40.4.5.11 `status_t CS42888_SetChannelMute ( cs42888_handle_t * handle, uint8_t channel, bool isMute )`

Parameters

|                |                                  |
|----------------|----------------------------------|
| <i>handle</i>  | CS42888 handle structure.        |
| <i>channel</i> | reference _cs42888_play_channel. |
| <i>isMute</i>  | true is mute, falase is unmute.  |

#### 40.4.5.12 `status_t CS42888_SetModule ( cs42888_handle_t * handle, cs42888_module_t module, bool isEnabled )`

Parameters

|                  |                            |
|------------------|----------------------------|
| <i>handle</i>    | CS42888 handle structure.  |
| <i>module</i>    | Module expected to enable. |
| <i>isEnabled</i> | Enable or disable moudles. |

#### 40.4.5.13 status\_t CS42888\_ConfigDataFormat ( *cs42888\_handle\_t \* handle, uint32\_t mclk, uint32\_t sample\_rate, uint32\_t bits* )

This function would configure the registers about the sample rate, bit depths.

Parameters

|                    |                                                                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>      | CS42888 handle structure pointer.                                                                                                           |
| <i>mclk</i>        | Master clock frequency of I2S.                                                                                                              |
| <i>sample_rate</i> | Sample rate of audio file running in CS42888. CS42888 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate. |
| <i>bits</i>        | Bit depth of audio file (CS42888 only supports 16bit, 20bit, 24bit and 32 bit in HW).                                                       |

#### 40.4.5.14 status\_t CS42888\_WriteReg ( *cs42888\_handle\_t \* handle, uint8\_t reg, uint8\_t val* )

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>handle</i> | CS42888 handle structure.               |
| <i>reg</i>    | The register address in CS42888.        |
| <i>val</i>    | Value needs to write into the register. |

#### 40.4.5.15 status\_t CS42888\_ReadReg ( *cs42888\_handle\_t \* handle, uint8\_t reg, uint8\_t \* val* )

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>handle</i> | CS42888 handle structure.        |
| <i>reg</i>    | The register address in CS42888. |
| <i>val</i>    | Value written to.                |

#### 40.4.5.16 status\_t CS42888\_ModifyReg ( *cs42888\_handle\_t \* handle, uint8\_t reg, uint8\_t mask, uint8\_t val* )

Parameters

|               |                                                                                  |
|---------------|----------------------------------------------------------------------------------|
| <i>handle</i> | CS42888 handle structure.                                                        |
| <i>reg</i>    | The register address in CS42888.                                                 |
| <i>mask</i>   | The mask code for the bits want to write. The bit you want to write should be 0. |
| <i>val</i>    | Value needs to write into the register.                                          |

## 40.4.6 CS42888 Adapter

### 40.4.6.1 Overview

The cs42888 adapter provides a codec unify control interface.

#### Macros

- `#define HAL_CODEC_CS42888_HANDLER_SIZE (CS42888_I2C_HANDLER_SIZE + 4)`  
*codec handler size*

#### Functions

- `status_t HAL_CODEC_CS42888_Init (void *handle, void *config)`  
*Codec initialization.*
- `status_t HAL_CODEC_CS42888_Deinit (void *handle)`  
*Codec de-initilization.*
- `status_t HAL_CODEC_CS42888_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `status_t HAL_CODEC_CS42888_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)`  
*set audio codec module volume.*
- `status_t HAL_CODEC_CS42888_SetMute (void *handle, uint32_t playChannel, bool isMute)`  
*set audio codec module mute.*
- `status_t HAL_CODEC_CS42888_SetPower (void *handle, uint32_t module, bool powerOn)`  
*set audio codec module power.*
- `status_t HAL_CODEC_CS42888_SetRecord (void *handle, uint32_t recordSource)`  
*codec set record source.*
- `status_t HAL_CODEC_CS42888_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)`  
*codec set record channel.*
- `status_t HAL_CODEC_CS42888_SetPlay (void *handle, uint32_t playSource)`  
*codec set play source.*
- `status_t HAL_CODEC_CS42888_ModuleControl (void *handle, uint32_t cmd, uint32_t data)`  
*codec module control.*
- `static status_t HAL_CODEC_Init (void *handle, void *config)`  
*Codec initilization.*
- `static status_t HAL_CODEC_Deinit (void *handle)`  
*Codec de-initilization.*
- `static status_t HAL_CODEC_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `static status_t HAL_CODEC_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)`  
*set audio codec module volume.*
- `static status_t HAL_CODEC_SetMute (void *handle, uint32_t playChannel, bool isMute)`  
*set audio codec module mute.*
- `static status_t HAL_CODEC_SetPower (void *handle, uint32_t module, bool powerOn)`

- static `status_t HAL_CODEC_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- static `status_t HAL_CODEC_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- static `status_t HAL_CODEC_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- static `status_t HAL_CODEC_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

#### 40.4.6.2 Function Documentation

##### 40.4.6.2.1 `status_t HAL_CODEC_CS42888_Init( void * handle, void * config )`

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

Returns

kStatus\_Success is success, else initial failed.

##### 40.4.6.2.2 `status_t HAL_CODEC_CS42888_Deinit( void * handle )`

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

##### 40.4.6.2.3 `status_t HAL_CODEC_CS42888_SetFormat( void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth )`

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

Returns

kStatus\_Success is success, else configure failed.

#### 40.4.6.2.4 status\_t HAL\_CODEC\_CS42888\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

Returns

kStatus\_Success is success, else configure failed.

#### 40.4.6.2.5 status\_t HAL\_CODEC\_CS42888\_SetMute ( void \* *handle*, uint32\_t *playChannel*, bool *isMute* )

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

Returns

kStatus\_Success is success, else configure failed.

#### 40.4.6.2.6 status\_t HAL\_CODEC\_CS42888\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* )

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.4.6.2.7 status\_t HAL\_CODEC\_CS42888\_SetRecord ( void \* *handle*, uint32\_t *recordSource* )

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.4.6.2.8 status\_t HAL\_CODEC\_CS42888\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* )

Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

Returns

kStatus\_Success is success, else configure failed.

#### 40.4.6.2.9 status\_t HAL\_CODEC\_CS42888\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.4.6.2.10 status\_t HAL\_CODEC\_CS42888\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* )

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.4.6.2.11 static status\_t HAL\_CODEC\_Init ( void \* *handle*, void \* *config* ) [inline], [static]

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

Returns

kStatus\_Success is success, else initial failed.

#### 40.4.6.2.12 static status\_t HAL\_CODEC\_Deinit ( void \* *handle* ) [inline], [static]

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

#### 40.4.6.2.13 static status\_t HAL\_CODEC\_SetFormat( void \* *handle*, uint32\_t *mclk*, uint32\_t *sampleRate*, uint32\_t *bitWidth* ) [inline], [static]

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

Returns

kStatus\_Success is success, else configure failed.

#### 40.4.6.2.14 static status\_t HAL\_CODEC\_SetVolume( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

Returns

kStatus\_Success is success, else configure failed.

#### 40.4.6.2.15 static status\_t HAL\_CODEC\_SetMute( void \* *handle*, uint32\_t *playChannel*, bool *isMute* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

Returns

kStatus\_Success is success, else configure failed.

#### 40.4.6.2.16 static status\_t HAL\_CODEC\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* ) [inline], [static]

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.4.6.2.17 static status\_t HAL\_CODEC\_SetRecord ( void \* *handle*, uint32\_t *recordSource* ) [inline], [static]

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.4.6.2.18 static status\_t HAL\_CODEC\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* ) [inline], [static]

Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |

Returns

kStatus\_Success is success, else configure failed.

**40.4.6.2.19 static status\_t HAL\_CODEC\_SetPlay ( void \* *handle*, uint32\_t *playSource* ) [inline], [static]**

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

**40.4.6.2.20 static status\_t HAL\_CODEC\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* ) [inline], [static]**

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

## 40.5 DA7212 Driver

### 40.5.1 Overview

The da7212 driver provides a codec control interface.

## Data Structures

- struct [da7212\\_pll\\_config\\_t](#)  
*da7212 pll configuration* [More...](#)
- struct [da7212\\_audio\\_format\\_t](#)  
*da7212 audio format* [More...](#)
- struct [da7212\\_config\\_t](#)  
*DA7212 configure structure.* [More...](#)
- struct [da7212\\_handle\\_t](#)  
*da7212 codec handler* [More...](#)

## Macros

- #define [DA7212\\_I2C\\_HANDLER\\_SIZE](#) CODEC\_I2C\_MASTER\_HANDLER\_SIZE  
*da7212 handle size*
- #define [DA7212\\_ADDRESS](#) (0x1A)  
*DA7212 I2C address.*
- #define [DA7212\\_HEADPHONE\\_MAX\\_VOLUME\\_VALUE](#) 0x3FU  
*da7212 volume setting range*

## Enumerations

- enum [da7212\\_Input\\_t](#) {  
 kDA7212\_Input\_AUX = 0x0,  
 kDA7212\_Input\_MIC1\_Dig,  
 kDA7212\_Input\_MIC1\_An,  
 kDA7212\_Input\_MIC2 }  
*DA7212 input source select.*
- enum [\\_da7212\\_play\\_channel](#) {  
 kDA7212\_HeadphoneLeft = 1U,  
 kDA7212\_HeadphoneRight = 2U,  
 kDA7212\_Speaker = 4U }  
*da7212 play channel*
- enum [da7212\\_Output\\_t](#) {  
 kDA7212\_Output\_HP = 0x0,  
 kDA7212\_Output\_SP }  
*DA7212 output device select.*

- enum \_da7212\_module {
 kDA7212\_ModuleADC,
 kDA7212\_ModuleDAC,
 kDA7212\_ModuleHeadphone,
 kDA7212\_ModuleSpeaker }
   
*DA7212 module.*
- enum da7212\_dac\_source\_t {
 kDA7212\_DACSourceADC = 0x0U,
 kDA7212\_DACSourceInputStream = 0x3U }
   
*DA7212 functionality.*
- enum da7212\_volume\_t {
 kDA7212\_DACGainMute = 0x7,
 kDA7212\_DACGainM72DB = 0x17,
 kDA7212\_DACGainM60DB = 0x1F,
 kDA7212\_DACGainM54DB = 0x27,
 kDA7212\_DACGainM48DB = 0x2F,
 kDA7212\_DACGainM42DB = 0x37,
 kDA7212\_DACGainM36DB = 0x3F,
 kDA7212\_DACGainM30DB = 0x47,
 kDA7212\_DACGainM24DB = 0x4F,
 kDA7212\_DACGainM18DB = 0x57,
 kDA7212\_DACGainM12DB = 0x5F,
 kDA7212\_DACGainM6DB = 0x67,
 kDA7212\_DACGain0DB = 0x6F,
 kDA7212\_DACGain6DB = 0x77,
 kDA7212\_DACGain12DB = 0x7F }
   
*DA7212 volume.*
- enum da7212\_protocol\_t {
 kDA7212\_BusI2S = 0x0,
 kDA7212\_BusLeftJustified,
 kDA7212\_BusRightJustified,
 kDA7212\_BusDSPMode }
   
*The audio data transfer protocol choice.*
- enum da7212\_sys\_clk\_source\_t {
 kDA7212\_SysClkSourceMCLK = 0U,
 kDA7212\_SysClkSourcePLL = 1U << 14 }
   
*da7212 system clock source*
- enum da7212\_pll\_clk\_source\_t { kDA7212\_PLLClkSourceMCLK = 0U }
   
*DA7212 pll clock source.*
- enum da7212\_pll\_out\_clk\_t {
 kDA7212\_PLLOutputClk11289600 = 11289600U,
 kDA7212\_PLLOutputClk12288000 = 12288000U }
   
*DA7212 output clock frequency.*
- enum da7212\_master\_bits\_t { }

```

kDA7212_MasterBits32PerFrame = 0U,
kDA7212_MasterBits64PerFrame = 1U,
kDA7212_MasterBits128PerFrame = 2U,
kDA7212_MasterBits256PerFrame = 3U }
 master mode bits per frame

```

## Functions

- `status_t DA7212_Init (da7212_handle_t *handle, da7212_config_t *codecConfig)`  
*DA7212 initialize function.*
- `status_t DA7212_ConfigAudioFormat (da7212_handle_t *handle, uint32_t masterClock_Hz, uint32_t sampleRate_Hz, uint32_t dataBits)`  
*Set DA7212 audio format.*
- `status_t DA7212_SetPLLConfig (da7212_handle_t *handle, da7212_pll_config_t *config)`  
*DA7212 set PLL configuration This function will enable the GPIO1 FLL clock output function, so user can see the generated fll output clock frequency from WM8904 GPIO1.*
- `void DA7212_ChangeHPVolume (da7212_handle_t *handle, da7212_volume_t volume)`  
*Set DA7212 playback volume.*
- `void DA7212_Mute (da7212_handle_t *handle, bool isMuted)`  
*Mute or unmute DA7212.*
- `void DA7212_ChangeInput (da7212_handle_t *handle, da7212_Input_t DA7212_Input)`  
*Set the input data source of DA7212.*
- `void DA7212_ChangeOutput (da7212_handle_t *handle, da7212_Output_t DA7212_Output)`  
*Set the output device of DA7212.*
- `status_t DA7212_SetChannelVolume (da7212_handle_t *handle, uint32_t channel, uint32_t volume)`  
*Set module volume.*
- `status_t DA7212_SetChannelMute (da7212_handle_t *handle, uint32_t channel, bool isMute)`  
*Set module mute.*
- `status_t DA7212_SetProtocol (da7212_handle_t *handle, da7212_protocol_t protocol)`  
*Set protocol for DA7212.*
- `status_t DA7212_SetMasterModeBits (da7212_handle_t *handle, uint32_t bitWidth)`  
*Set master mode bits per frame for DA7212.*
- `status_t DA7212_WriteRegister (da7212_handle_t *handle, uint8_t u8Register, uint8_t u8RegisterData)`  
*Write a register for DA7212.*
- `status_t DA7212_ReadRegister (da7212_handle_t *handle, uint8_t u8Register, uint8_t *pu8RegisterData)`  
*Get a register value of DA7212.*
- `status_t DA7212_Deinit (da7212_handle_t *handle)`  
*Deinit DA7212.*

## Driver version

- `#define FSL_DA7212_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))`  
*CLOCK driver version 2.2.2.*

## 40.5.2 Data Structure Documentation

### 40.5.2.1 struct da7212\_pll\_config\_t

#### Data Fields

- `da7212_pll_clk_source_t source`  
*pll reference clock source*
- `uint32_t refClock_HZ`  
*pll reference clock frequency*
- `da7212_pll_out_clk_t outputClock_HZ`  
*pll output clock frequency*

### 40.5.2.2 struct da7212\_audio\_format\_t

#### Data Fields

- `uint32_t mclk_HZ`  
*master clock frequency*
- `uint32_t sampleRate`  
*sample rate*
- `uint32_t bitWidth`  
*bit width*
- `bool isBclkInvert`  
*bit clock invertet*

### 40.5.2.3 struct da7212\_config\_t

#### Data Fields

- `bool isMaster`  
*If DA7212 is master, true means master, false means slave.*
- `da7212_protocol_t protocol`  
*Audio bus format, can be I2S, LJ, RJ or DSP mode.*
- `da7212_dac_source_t dacSource`  
*DA7212 data source.*
- `da7212_audio_format_t format`  
*audio format*
- `uint8_t slaveAddress`  
*device address*
- `codec_i2c_config_t i2cConfig`  
*i2c configuration*
- `da7212_sys_clk_source_t sysClkSource`  
*system clock source*
- `da7212_pll_config_t * pll`  
*pll configuration*

#### Field Documentation

- (1) `bool da7212_config_t::isMaster`
- (2) `da7212_protocol_t da7212_config_t::protocol`
- (3) `da7212_dac_source_t da7212_config_t::dacSource`

#### 40.5.2.4 struct da7212\_handle\_t

##### Data Fields

- `da7212_config_t * config`  
*da7212 config pointer*
- `uint8_t i2cHandle [DA7212_I2C_HANDLER_SIZE]`  
*i2c handle*

#### 40.5.3 Macro Definition Documentation

##### 40.5.3.1 #define FSL\_DA7212\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 2))

#### 40.5.4 Enumeration Type Documentation

##### 40.5.4.1 enum da7212\_Input\_t

Enumerator

- kDA7212\_Input\_AUX* Input from AUX.
- kDA7212\_Input\_MIC1\_Dig* Input from MIC1 Digital.
- kDA7212\_Input\_MIC1\_An* Input from Mic1 Analog.
- kDA7212\_Input\_MIC2* Input from MIC2.

##### 40.5.4.2 enum \_da7212\_play\_channel

Enumerator

- kDA7212\_HeadphoneLeft* headphone left
- kDA7212\_HeadphoneRight* headphone right
- kDA7212\_Speaker* speaker channel

##### 40.5.4.3 enum da7212\_Output\_t

Enumerator

- kDA7212\_Output\_HP* Output to headphone.
- kDA7212\_Output\_SP* Output to speaker.

#### 40.5.4.4 enum \_da7212\_module

Enumerator

*kDA7212\_ModuleADC* module ADC  
*kDA7212\_ModuleDAC* module DAC  
*kDA7212\_ModuleHeadphone* module headphone  
*kDA7212\_ModuleSpeaker* module speaker

#### 40.5.4.5 enum da7212\_dac\_source\_t

Enumerator

*kDA7212\_DACSourceADC* DAC source from ADC.  
*kDA7212\_DACSourceInputStream* DAC source from.

#### 40.5.4.6 enum da7212\_volume\_t

Enumerator

*kDA7212\_DACGainMute* Mute DAC.  
*kDA7212\_DACGainM72DB* DAC volume -72db.  
*kDA7212\_DACGainM60DB* DAC volume -60db.  
*kDA7212\_DACGainM54DB* DAC volume -54db.  
*kDA7212\_DACGainM48DB* DAC volume -48db.  
*kDA7212\_DACGainM42DB* DAC volume -42db.  
*kDA7212\_DACGainM36DB* DAC volume -36db.  
*kDA7212\_DACGainM30DB* DAC volume -30db.  
*kDA7212\_DACGainM24DB* DAC volume -24db.  
*kDA7212\_DACGainM18DB* DAC volume -18db.  
*kDA7212\_DACGainM12DB* DAC volume -12db.  
*kDA7212\_DACGainM6DB* DAC volume -6db.  
*kDA7212\_DACGain0DB* DAC volume +0db.  
*kDA7212\_DACGain6DB* DAC volume +6db.  
*kDA7212\_DACGain12DB* DAC volume +12db.

#### 40.5.4.7 enum da7212\_protocol\_t

Enumerator

*kDA7212\_BusI2S* I2S Type.  
*kDA7212\_BusLeftJustified* Left justified.  
*kDA7212\_BusRightJustified* Right Justified.  
*kDA7212\_BusDSPMode* DSP mode.

#### 40.5.4.8 enum da7212\_sys\_clk\_source\_t

Enumerator

*kDA7212\_SysClkSourceMCLK* da7212 system clock soure from MCLK

*kDA7212\_SysClkSourcePLL* da7212 system clock soure from pLL

#### 40.5.4.9 enum da7212\_pll\_clk\_source\_t

Enumerator

*kDA7212\_PLLClkSourceMCLK* DA7212 PLL clock source from MCLK.

#### 40.5.4.10 enum da7212\_pll\_out\_clk\_t

Enumerator

*kDA7212\_PLLOutputClk11289600* output 112896000U

*kDA7212\_PLLOutputClk12288000* output 12288000U

#### 40.5.4.11 enum da7212\_master\_bits\_t

Enumerator

*kDA7212\_MasterBits32PerFrame* master mode bits32 per frame

*kDA7212\_MasterBits64PerFrame* master mode bits64 per frame

*kDA7212\_MasterBits128PerFrame* master mode bits128 per frame

*kDA7212\_MasterBits256PerFrame* master mode bits256 per frame

### 40.5.5 Function Documentation

#### 40.5.5.1 status\_t DA7212\_Init ( da7212\_handle\_t \* *handle*, da7212\_config\_t \* *codecConfig* )

Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | DA7212 handle pointer. |
|---------------|------------------------|

|                    |                                                                                                                                                                                                                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>codecConfig</i> | Codec configure structure. This parameter can be NULL, if NULL, set as default settings. The default setting:<br><br><pre>* sgtl_init_t codec_config * codec_config.route = kDA7212_RoutePlayback * codec_config.bus = <b>kDA7212_BusI2S</b> * codec_config.isMaster = <b>false</b> *</pre> |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

#### 40.5.5.2 status\_t DA7212\_ConfigAudioFormat ( *da7212\_handle\_t \* handle, uint32\_t masterClock\_Hz, uint32\_t sampleRate\_Hz, uint32\_t dataBits* )

Parameters

|                       |                                                                                                                                                                                                                                        |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>         | DA7212 handle pointer.                                                                                                                                                                                                                 |
| <i>masterClock_Hz</i> | Master clock frequency in Hz. If DA7212 is slave, use the frequency of master, if DA7212 as master, it should be 1228000 while sample rate frequency is 8k/12K/16-K/24K/32K/48K/96K, 11289600 whie sample rate is 11.025K/22.05K/44.1K |
| <i>sampleRate_Hz</i>  | Sample rate frequency in Hz.                                                                                                                                                                                                           |
| <i>dataBits</i>       | How many bits in a word of a audio frame, DA7212 only supports 16/20/24/32 bits.                                                                                                                                                       |

#### 40.5.5.3 status\_t DA7212\_SetPLLConfig ( *da7212\_handle\_t \* handle, da7212\_pll\_config\_t \* config* )

Parameters

|               |                            |
|---------------|----------------------------|
| <i>handle</i> | DA7212 handler pointer.    |
| <i>config</i> | PLL configuration pointer. |

#### 40.5.5.4 void DA7212\_ChangeHPVolume ( *da7212\_handle\_t \* handle, da7212\_volume\_t volume* )

Parameters

|               |                         |
|---------------|-------------------------|
| <i>handle</i> | DA7212 handle pointer.  |
| <i>volume</i> | The volume of playback. |

**40.5.5.5 void DA7212\_Mute ( da7212\_handle\_t \* *handle*, bool *isMuted* )**

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>handle</i>  | DA7212 handle pointer.               |
| <i>isMuted</i> | True means mute, false means unmute. |

**40.5.5.6 void DA7212\_ChangeInput ( da7212\_handle\_t \* *handle*, da7212\_Input\_t *DA7212\_Input* )**

Parameters

|                     |                        |
|---------------------|------------------------|
| <i>handle</i>       | DA7212 handle pointer. |
| <i>DA7212_Input</i> | Input data source.     |

**40.5.5.7 void DA7212\_ChangeOutput ( da7212\_handle\_t \* *handle*, da7212\_Output\_t *DA7212\_Output* )**

Parameters

|                      |                          |
|----------------------|--------------------------|
| <i>handle</i>        | DA7212 handle pointer.   |
| <i>DA7212_Output</i> | Output device of DA7212. |

**40.5.5.8 status\_t DA7212\_SetChannelVolume ( da7212\_handle\_t \* *handle*, uint32\_t *channel*, uint32\_t *volume* )**

Parameters

|                |                                                    |
|----------------|----------------------------------------------------|
| <i>handle</i>  | DA7212 handle pointer.                             |
| <i>channel</i> | shoule be a value of _da7212_channel.              |
| <i>volume</i>  | volume range 0 - 0x3F mapped to range -57dB - 6dB. |

#### 40.5.5.9 status\_t DA7212\_SetChannelMute ( *da7212\_handle\_t \* handle, uint32\_t channel, bool isMute* )

Parameters

|                |                                       |
|----------------|---------------------------------------|
| <i>handle</i>  | DA7212 handle pointer.                |
| <i>channel</i> | shoule be a value of _da7212_channel. |
| <i>isMute</i>  | true is mute, false is unmute.        |

#### 40.5.5.10 status\_t DA7212\_SetProtocol ( *da7212\_handle\_t \* handle, da7212\_protocol\_t protocol* )

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>handle</i>   | DA7212 handle pointer. |
| <i>protocol</i> | da7212_protocol_t.     |

#### 40.5.5.11 status\_t DA7212\_SetMasterModeBits ( *da7212\_handle\_t \* handle, uint32\_t bitWidth* )

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>handle</i>   | DA7212 handle pointer. |
| <i>bitWidth</i> | audio data bitwidth.   |

#### 40.5.5.12 status\_t DA7212\_WriteRegister ( *da7212\_handle\_t \* handle, uint8\_t u8Register, uint8\_t u8RegisterData* )

Parameters

|                       |                                        |
|-----------------------|----------------------------------------|
| <i>handle</i>         | DA7212 handle pointer.                 |
| <i>u8Register</i>     | DA7212 register address to be written. |
| <i>u8RegisterData</i> | Data to be written into register       |

#### 40.5.5.13 status\_t DA7212\_ReadRegister ( da7212\_handle\_t \* *handle*, uint8\_t *u8Register*, uint8\_t \* *pu8RegisterData* )

Parameters

|                        |                                                |
|------------------------|------------------------------------------------|
| <i>handle</i>          | DA7212 handle pointer.                         |
| <i>u8Register</i>      | DA7212 register address to be read.            |
| <i>pu8RegisterData</i> | Pointer where the read out value to be stored. |

#### 40.5.5.14 status\_t DA7212\_Deinit ( da7212\_handle\_t \* *handle* )

Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | DA7212 handle pointer. |
|---------------|------------------------|

## 40.5.6 DA7212 Adapter

### 40.5.6.1 Overview

The da7212 adapter provides a codec unify control interface.

#### Macros

- #define `HAL_CODEC_DA7212_HANDLER_SIZE` (`DA7212_I2C_HANDLER_SIZE + 4`)  
*codec handler size*

#### Functions

- `status_t HAL_CODEC_DA7212_Init` (void \*handle, void \*config)  
*Codec initialization.*
- `status_t HAL_CODEC_DA7212_Deinit` (void \*handle)  
*Codec de-initilization.*
- `status_t HAL_CODEC_DA7212_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- `status_t HAL_CODEC_DA7212_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- `status_t HAL_CODEC_DA7212_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- `status_t HAL_CODEC_DA7212_SetPower` (void \*handle, uint32\_t module, bool powerOn)  
*set audio codec module power.*
- `status_t HAL_CODEC_DA7212_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- `status_t HAL_CODEC_DA7212_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- `status_t HAL_CODEC_DA7212_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- `status_t HAL_CODEC_DA7212_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*
- static `status_t HAL_CODEC_Init` (void \*handle, void \*config)  
*Codec initilization.*
- static `status_t HAL_CODEC_Deinit` (void \*handle)  
*Codec de-initilization.*
- static `status_t HAL_CODEC_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- static `status_t HAL_CODEC_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- static `status_t HAL_CODEC_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- static `status_t HAL_CODEC_SetPower` (void \*handle, uint32\_t module, bool powerOn)

- static `status_t HAL_CODEC_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- static `status_t HAL_CODEC_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- static `status_t HAL_CODEC_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- static `status_t HAL_CODEC_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

#### 40.5.6.2 Function Documentation

##### 40.5.6.2.1 `status_t HAL_CODEC_DA7212_Init( void * handle, void * config )`

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

Returns

kStatus\_Success is success, else initial failed.

##### 40.5.6.2.2 `status_t HAL_CODEC_DA7212_Deinit( void * handle )`

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

##### 40.5.6.2.3 `status_t HAL_CODEC_DA7212_SetFormat( void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth )`

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

Returns

kStatus\_Success is success, else configure failed.

#### 40.5.6.2.4 status\_t HAL\_CODEC\_DA7212\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

Returns

kStatus\_Success is success, else configure failed.

#### 40.5.6.2.5 status\_t HAL\_CODEC\_DA7212\_SetMute ( void \* *handle*, uint32\_t *playChannel*, bool *isMute* )

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

Returns

kStatus\_Success is success, else configure failed.

#### 40.5.6.2.6 status\_t HAL\_CODEC\_DA7212\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* )

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.5.6.2.7 status\_t HAL\_CODEC\_DA7212\_SetRecord ( void \* *handle*, uint32\_t *recordSource* )

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.5.6.2.8 status\_t HAL\_CODEC\_DA7212\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* )

Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

Returns

kStatus\_Success is success, else configure failed.

#### 40.5.6.2.9 status\_t HAL\_CODEC\_DA7212\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.5.6.2.10 status\_t HAL\_CODEC\_DA7212\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* )

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.5.6.2.11 static status\_t HAL\_CODEC\_Init ( void \* *handle*, void \* *config* ) [inline], [static]

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

Returns

kStatus\_Success is success, else initial failed.

#### 40.5.6.2.12 static status\_t HAL\_CODEC\_Deinit ( void \* *handle* ) [inline], [static]

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

#### 40.5.6.2.13 static status\_t HAL\_CODEC\_SetFormat( void \* *handle*, uint32\_t *mclk*, uint32\_t *sampleRate*, uint32\_t *bitWidth* ) [inline], [static]

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

Returns

kStatus\_Success is success, else configure failed.

#### 40.5.6.2.14 static status\_t HAL\_CODEC\_SetVolume( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

Returns

kStatus\_Success is success, else configure failed.

#### 40.5.6.2.15 static status\_t HAL\_CODEC\_SetMute( void \* *handle*, uint32\_t *playChannel*, bool *isMute* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

Returns

kStatus\_Success is success, else configure failed.

#### 40.5.6.2.16 static status\_t HAL\_CODEC\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* ) [inline], [static]

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.5.6.2.17 static status\_t HAL\_CODEC\_SetRecord ( void \* *handle*, uint32\_t *recordSource* ) [inline], [static]

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.5.6.2.18 static status\_t HAL\_CODEC\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* ) [inline], [static]

Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

Returns

kStatus\_Success is success, else configure failed.

**40.5.6.2.19 static status\_t HAL\_CODEC\_SetPlay ( void \* *handle*, uint32\_t *playSource* ) [inline], [static]**

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

**40.5.6.2.20 static status\_t HAL\_CODEC\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* ) [inline], [static]**

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

## 40.6 SGTL5000 Driver

### 40.6.1 Overview

The sgtl5000 driver provides a codec control interface.

### Data Structures

- struct `sgtl_audio_format_t`  
*Audio format configuration. [More...](#)*
- struct `sgtl_config_t`  
*Initialize structure of sgtl5000. [More...](#)*
- struct `sgtl_handle_t`  
*SGTL codec handler. [More...](#)*

### Macros

- #define `CHIP_ID` 0x0000U  
*Define the register address of sgtl5000.*
- #define `SGTL5000_HEADPHONE_MAX_VOLUME_VALUE` 0x7FU  
*SGTL5000 volume setting range.*
- #define `SGTL5000_I2C_ADDR` 0x0A  
*SGTL5000 I2C address.*
- #define `SGTL_I2C_HANDLER_SIZE` CODEC\_I2C\_MASTER\_HANDLER\_SIZE  
*sgtl handle size*
- #define `SGTL_I2C_BITRATE` 100000U  
*sgtl i2c baudrate*

### Enumerations

- enum `sgtl_module_t` {
   
`kSGTL_ModuleADC` = 0x0,  
`kSGTL_ModuleDAC`,  
`kSGTL_ModuleDAP`,  
`kSGTL_ModuleHP`,  
`kSGTL_ModuleI2SIN`,  
`kSGTL_ModuleI2SOUT`,  
`kSGTL_ModuleLineIn`,  
`kSGTL_ModuleLineOut`,  
`kSGTL_ModuleMicin` }
- Modules in Sglt5000 board.*
- enum `sgtl_route_t` {

```

kSGTL_RouteBypass = 0x0,
kSGTL_RoutePlayback,
kSGTL_RoutePlaybackandRecord,
kSGTL_RoutePlaybackwithDAP,
kSGTL_RoutePlaybackwithDAPandRecord,
kSGTL_RouteRecord }

Sgtl5000 data route.
• enum sgtl_protocol_t {
 kSGTL_BusI2S = 0x0,
 kSGTL_BusLeftJustified,
 kSGTL_BusRightJustified,
 kSGTL_BusPCMA,
 kSGTL_BusPCMB }

The audio data transfer protocol choice.
• enum {
 kSGTL_HeadphoneLeft = 0,
 kSGTL_HeadphoneRight = 1,
 kSGTL_LineoutLeft = 2,
 kSGTL_LineoutRight = 3 }

sgtl play channel
• enum {
 kSGTL_RecordSourceLineIn = 0U,
 kSGTL_RecordSourceMic = 1U }

sgtl record source _sgtl_record_source
• enum {
 kSGTL_PlaySourceLineIn = 0U,
 kSGTL_PlaySourceDAC = 1U }

sgtl play source _stgl_play_source
• enum sgtl_sclk_edge_t {
 kSGTL_SclkValidEdgeRising = 0U,
 kSGTL_SclkValidEdgeFailling = 1U }

SGTL SCLK valid edge.

```

## Functions

- `status_t SGTL_Init (sgtl_handle_t *handle, sgtl_config_t *config)`  
*sgtl5000 initialize function.*
- `status_t SGTL_SetDataRoute (sgtl_handle_t *handle, sgtl_route_t route)`  
*Set audio data route in sgtl5000.*
- `status_t SGTL_SetProtocol (sgtl_handle_t *handle, sgtl_protocol_t protocol)`  
*Set the audio transfer protocol.*
- `void SGTL_SetMasterSlave (sgtl_handle_t *handle, bool master)`  
*Set sgtl5000 as master or slave.*
- `status_t SGTL_SetVolume (sgtl_handle_t *handle, sgtl_module_t module, uint32_t volume)`  
*Set the volume of different modules in sgtl5000.*
- `uint32_t SGTL_GetVolume (sgtl_handle_t *handle, sgtl_module_t module)`  
*Get the volume of different modules in sgtl5000.*

- `status_t SGTL_SetMute (sgtl_handle_t *handle, sgtl_module_t module, bool mute)`  
*Mute/unmute modules in sgtl5000.*
- `status_t SGTL_EnableModule (sgtl_handle_t *handle, sgtl_module_t module)`  
*Enable expected devices.*
- `status_t SGTL_DisableModule (sgtl_handle_t *handle, sgtl_module_t module)`  
*Disable expected devices.*
- `status_t SGTL_Deinit (sgtl_handle_t *handle)`  
*Deinit the sgtl5000 codec.*
- `status_t SGTL_ConfigDataFormat (sgtl_handle_t *handle, uint32_t mclk, uint32_t sample_rate, uint32_t bits)`  
*Configure the data format of audio data.*
- `status_t SGTL_SetPlay (sgtl_handle_t *handle, uint32_t playSource)`  
*select SGTL codec play source.*
- `status_t SGTL_SetRecord (sgtl_handle_t *handle, uint32_t recordSource)`  
*select SGTL codec record source.*
- `status_t SGTL_WriteReg (sgtl_handle_t *handle, uint16_t reg, uint16_t val)`  
*Write register to sgtl using I2C.*
- `status_t SGTL_ReadReg (sgtl_handle_t *handle, uint16_t reg, uint16_t *val)`  
*Read register from sgtl using I2C.*
- `status_t SGTL_ModifyReg (sgtl_handle_t *handle, uint16_t reg, uint16_t clr_mask, uint16_t val)`  
*Modify some bits in the register using I2C.*

## Driver version

- `#define FSL_SGTL5000_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`  
*CLOCK driver version 2.1.1.*

### 40.6.2 Data Structure Documentation

#### 40.6.2.1 struct sgtl\_audio\_format\_t

##### Data Fields

- `uint32_t mclk_HZ`  
*master clock*
- `uint32_t sampleRate`  
*Sample rate.*
- `uint32_t bitWidth`  
*Bit width.*
- `sgtl_sclk_edge_t sclkEdge`  
*sclk valid edge*

#### 40.6.2.2 struct sgtl\_config\_t

##### Data Fields

- `sgtl_route_t route`

- *Audio data route.*
- `sgtl_protocol_t bus`  
*Audio transfer protocol.*
- `bool master_slave`  
*Master or slave.*
- `sgtl_audio_format_t format`  
*audio format*
- `uint8_t slaveAddress`  
*code device slave address*
- `codec_i2c_config_t i2cConfig`  
*i2c bus configuration*

## Field Documentation

- (1) `sgtl_route_t sgtl_config_t::route`
- (2) `bool sgtl_config_t::master_slave`

True means master, false means slave.

### 40.6.2.3 struct sgtl\_handle\_t

#### Data Fields

- `sgtl_config_t * config`  
*sgtl config pointer*
- `uint8_t i2cHandle [SGTL_I2C_HANDLER_SIZE]`  
*i2c handle*

### 40.6.3 Macro Definition Documentation

#### 40.6.3.1 #define FSL\_SGTL5000\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))

#### 40.6.3.2 #define CHIP\_ID 0x0000U

#### 40.6.3.3 #define SGTL5000\_I2C\_ADDR 0x0A

### 40.6.4 Enumeration Type Documentation

#### 40.6.4.1 enum sgtl\_module\_t

Enumerator

- `kSGTL_ModuleADC` ADC module in SGTL5000.
- `kSGTL_ModuleDAC` DAC module in SGTL5000.
- `kSGTL_ModuleDAP` DAP module in SGTL5000.
- `kSGTL_ModuleHP` Headphone module in SGTL5000.

*kSGTL\_ModuleI2SIN* I2S-IN module in SGTL5000.  
*kSGTL\_ModuleI2SOUT* I2S-OUT module in SGTL5000.  
*kSGTL\_ModuleLineIn* Line-in moudle in SGTL5000.  
*kSGTL\_ModuleLineOut* Line-out module in SGTL5000.  
*kSGTL\_ModuleMicin* Micphone module in SGTL5000.

#### 40.6.4.2 enum sgtl\_route\_t

Note

Only provide some typical data route, not all route listed. Users cannot combine any routes, once a new route is set, the previous one would be replaced.

Enumerator

*kSGTL\_RouteBypass* LINEIN->Headphone.  
*kSGTL\_RoutePlayback* I2SIN->DAC->Headphone.  
*kSGTL\_RoutePlaybackandRecord* I2SIN->DAC->Headphone, LINEIN->ADC->I2SOUT.  
*kSGTL\_RoutePlaybackwithDAP* I2SIN->DAP->DAC->Headphone.  
*kSGTL\_RoutePlaybackwithDAPandRecord* I2SIN->DAP->DAC->HP, LINEIN->ADC->I2SOUT.  
*kSGTL\_RouteRecord* LINEIN->ADC->I2SOUT.

#### 40.6.4.3 enum sgtl\_protocol\_t

Sgtl5000 only supports I2S format and PCM format.

Enumerator

*kSGTL\_BusI2S* I2S Type.  
*kSGTL\_BusLeftJustified* Left justified.  
*kSGTL\_BusRightJustified* Right Justified.  
*kSGTL\_BusPCMA* PCMA.  
*kSGTL\_BusPCMB* PCMB.

#### 40.6.4.4 anonymous enum

Enumerator

*kSGTL\_HeadphoneLeft* headphone left channel  
*kSGTL\_HeadphoneRight* headphone right channel  
*kSGTL\_LineoutLeft* lineout left channel  
*kSGTL\_LineoutRight* lineout right channel

#### 40.6.4.5 anonymous enum

Enumerator

*kSGTL\_RecordSourceLineIn* record source line in  
*kSGTL\_RecordSourceMic* record source single end

#### 40.6.4.6 anonymous enum

Enumerator

*kSGTL\_PlaySourceLineIn* play source line in  
*kSGTL\_PlaySourceDAC* play source line in

#### 40.6.4.7 enum sgtl\_sclk\_edge\_t

Enumerator

*kSGTL\_SclkValidEdgeRising* SCLK valid edge.  
*kSGTL\_SclkValidEdgeFailling* SCLK failling edge.

### 40.6.5 Function Documentation

#### 40.6.5.1 status\_t SGTL\_Init( sgtl\_handle\_t \* handle, sgtl\_config\_t \* config )

This function calls SGTL\_I2CInit(), and in this function, some configurations are fixed. The second parameter can be NULL. If users want to change the SGTL5000 settings, a configure structure should be prepared.

Note

If the codec\_config is NULL, it would initialize sgtl5000 using default settings. The default setting:

```
* sgtl_init_t codec_config
* codec_config.route = kSGTL_RoutePlaybackandRecord
* codec_config.bus = kSGTL_BusI2S
* codec_config.master = slave
*
```

Parameters

---

|               |                                                                                                             |
|---------------|-------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | Sgtl5000 handle structure.                                                                                  |
| <i>config</i> | sgtl5000 configuration structure. If this pointer equals to NULL, it means using the default configuration. |

Returns

Initialization status

#### 40.6.5.2 status\_t SGTL\_SetDataRoute ( sgtl\_handle\_t \* *handle*, sgtl\_route\_t *route* )

This function would set the data route according to route. The route cannot be combined, as all route would enable different modules.

Note

If a new route is set, the previous route would not work.

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>handle</i> | Sgtl5000 handle structure.    |
| <i>route</i>  | Audio data route in sgtl5000. |

#### 40.6.5.3 status\_t SGTL\_SetProtocol ( sgtl\_handle\_t \* *handle*, sgtl\_protocol\_t *protocol* )

Sgtl5000 only supports I2S, I2S left, I2S right, PCM A, PCM B format.

Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>handle</i>   | Sgtl5000 handle structure.    |
| <i>protocol</i> | Audio data transfer protocol. |

#### 40.6.5.4 void SGTL\_SetMasterSlave ( sgtl\_handle\_t \* *handle*, bool *master* )

Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>handle</i> | Sgtl5000 handle structure.             |
| <i>master</i> | 1 represent master, 0 represent slave. |

#### 40.6.5.5 status\_t SGTL\_SetVolume ( *sgtl\_handle\_t \* handle*, *sgtl\_module\_t module*, *uint32\_t volume* )

This function would set the volume of sgtl5000 modules. This interface set module volume. The function assume that left channel and right channel has the same volume.

kSGTL\_ModuleADC volume range: 0 - 0xF, 0dB - 22.5dB  
kSGTL\_ModuleDAC volume range: 0x3C - 0xF0, 0dB - -90dB  
kSGTL\_ModuleHP volume range: 0 - 0x7F, 12dB - -51.5dB  
kSGTL\_ModuleLineOut volume range: 0 - 0x1F, 0.5dB steps

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>handle</i> | Sgtl5000 handle structure.                                             |
| <i>module</i> | Sgtl5000 module, such as DAC, ADC and etc.                             |
| <i>volume</i> | Volume value need to be set. The value is the exact value in register. |

#### 40.6.5.6 uint32\_t SGTL\_GetVolume ( *sgtl\_handle\_t \* handle*, *sgtl\_module\_t module* )

This function gets the volume of sgtl5000 modules. This interface get DAC module volume. The function assume that left channel and right channel has the same volume.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>handle</i> | Sgtl5000 handle structure.                 |
| <i>module</i> | Sgtl5000 module, such as DAC, ADC and etc. |

Returns

Module value, the value is exact value in register.

#### 40.6.5.7 status\_t SGTL\_SetMute ( *sgtl\_handle\_t \* handle*, *sgtl\_module\_t module*, *bool mute* )

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>handle</i> | Sgtl5000 handle structure.                 |
| <i>module</i> | Sgtl5000 module, such as DAC, ADC and etc. |
| <i>mute</i>   | True means mute, and false means unmute.   |

#### 40.6.5.8 status\_t SGTL\_EnableModule ( *sgtl\_handle\_t \* handle*, *sgtl\_module\_t module* )

Parameters

|               |                            |
|---------------|----------------------------|
| <i>handle</i> | Sgtl5000 handle structure. |
| <i>module</i> | Module expected to enable. |

#### 40.6.5.9 status\_t SGTL\_DisableModule ( *sgtl\_handle\_t \* handle*, *sgtl\_module\_t module* )

Parameters

|               |                            |
|---------------|----------------------------|
| <i>handle</i> | Sgtl5000 handle structure. |
| <i>module</i> | Module expected to enable. |

#### 40.6.5.10 status\_t SGTL\_Deinit ( *sgtl\_handle\_t \* handle* )

Shut down Sgtl5000 modules.

Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>handle</i> | Sgtl5000 handle structure pointer. |
|---------------|------------------------------------|

#### 40.6.5.11 status\_t SGTL\_ConfigDataFormat ( *sgtl\_handle\_t \* handle*, *uint32\_t mclk*, *uint32\_t sample\_rate*, *uint32\_t bits* )

This function would configure the registers about the sample rate, bit depths.

Parameters

|                    |                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>      | Sgtl5000 handle structure pointer.                                                                                                            |
| <i>mclk</i>        | Master clock frequency of I2S.                                                                                                                |
| <i>sample_rate</i> | Sample rate of audio file running in sgtl5000. Sgtl5000 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate. |
| <i>bits</i>        | Bit depth of audio file (Sgtl5000 only supports 16bit, 20bit, 24bit and 32 bit in HW).                                                        |

#### 40.6.5.12 status\_t SGTL\_SetPlay ( *sgtl\_handle\_t \* handle, uint32\_t playSource* )

Parameters

|                   |                                                 |
|-------------------|-------------------------------------------------|
| <i>handle</i>     | Sgtl5000 handle structure pointer.              |
| <i>playSource</i> | play source value, reference _sgtl_play_source. |

Returns

kStatus\_Success, else failed.

#### 40.6.5.13 status\_t SGTL\_SetRecord ( *sgtl\_handle\_t \* handle, uint32\_t recordSource* )

Parameters

|                     |                                                     |
|---------------------|-----------------------------------------------------|
| <i>handle</i>       | Sgtl5000 handle structure pointer.                  |
| <i>recordSource</i> | record source value, reference _sgtl_record_source. |

Returns

kStatus\_Success, else failed.

#### 40.6.5.14 status\_t SGTL\_WriteReg ( *sgtl\_handle\_t \* handle, uint16\_t reg, uint16\_t val* )

Parameters

|               |                            |
|---------------|----------------------------|
| <i>handle</i> | Sgtl5000 handle structure. |
|---------------|----------------------------|

|            |                                         |
|------------|-----------------------------------------|
| <i>reg</i> | The register address in sgtl.           |
| <i>val</i> | Value needs to write into the register. |

#### 40.6.5.15 status\_t SGTL\_ReadReg ( *sgtl\_handle\_t \* handle, uint16\_t reg, uint16\_t \* val* )

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>handle</i> | Sgtl5000 handle structure.    |
| <i>reg</i>    | The register address in sgtl. |
| <i>val</i>    | Value written to.             |

#### 40.6.5.16 status\_t SGTL\_ModifyReg ( *sgtl\_handle\_t \* handle, uint16\_t reg, uint16\_t clr\_mask, uint16\_t val* )

Parameters

|                 |                                                                                  |
|-----------------|----------------------------------------------------------------------------------|
| <i>handle</i>   | Sgtl5000 handle structure.                                                       |
| <i>reg</i>      | The register address in sgtl.                                                    |
| <i>clr_mask</i> | The mask code for the bits want to write. The bit you want to write should be 0. |
| <i>val</i>      | Value needs to write into the register.                                          |

## 40.6.6 SGTL5000 Adapter

### 40.6.6.1 Overview

The sgtl5000 adapter provides a codec unify control interface.

#### Macros

- `#define HAL_CODEC_SGTL_HANDLER_SIZE (SGTL_I2C_HANDLER_SIZE + 4)`  
*codec handler size*

#### Functions

- `status_t HAL_CODEC_SGTL5000_Init (void *handle, void *config)`  
*Codec initialization.*
- `status_t HAL_CODEC_SGTL5000_Deinit (void *handle)`  
*Codec de-initilization.*
- `status_t HAL_CODEC_SGTL5000_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `status_t HAL_CODEC_SGTL5000_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)`  
*set audio codec module volume.*
- `status_t HAL_CODEC_SGTL5000_SetMute (void *handle, uint32_t playChannel, bool isMute)`  
*set audio codec module mute.*
- `status_t HAL_CODEC_SGTL5000_SetPower (void *handle, uint32_t module, bool powerOn)`  
*set audio codec module power.*
- `status_t HAL_CODEC_SGTL5000_SetRecord (void *handle, uint32_t recordSource)`  
*codec set record source.*
- `status_t HAL_CODEC_SGTL5000_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)`  
*codec set record channel.*
- `status_t HAL_CODEC_SGTL5000_SetPlay (void *handle, uint32_t playSource)`  
*codec set play source.*
- `status_t HAL_CODEC_SGTL5000_ModuleControl (void *handle, uint32_t cmd, uint32_t data)`  
*codec module control.*
- `static status_t HAL_CODEC_Init (void *handle, void *config)`  
*Codec initilization.*
- `static status_t HAL_CODEC_Deinit (void *handle)`  
*Codec de-initilization.*
- `static status_t HAL_CODEC_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `static status_t HAL_CODEC_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)`  
*set audio codec module volume.*
- `static status_t HAL_CODEC_SetMute (void *handle, uint32_t playChannel, bool isMute)`  
*set audio codec module mute.*
- `static status_t HAL_CODEC_SetPower (void *handle, uint32_t module, bool powerOn)`

- static `status_t HAL_CODEC_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- static `status_t HAL_CODEC_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- static `status_t HAL_CODEC_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- static `status_t HAL_CODEC_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

#### 40.6.6.2 Function Documentation

##### 40.6.6.2.1 `status_t HAL_CODEC_SGTL5000_Init( void * handle, void * config )`

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

Returns

`kStatus_Success` is success, else initial failed.

##### 40.6.6.2.2 `status_t HAL_CODEC_SGTL5000_Deinit( void * handle )`

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

`kStatus_Success` is success, else de-initial failed.

##### 40.6.6.2.3 `status_t HAL_CODEC_SGTL5000_SetFormat( void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth )`

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

Returns

kStatus\_Success is success, else configure failed.

#### 40.6.6.2.4 status\_t HAL\_CODEC\_SGTL5000\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

Returns

kStatus\_Success is success, else configure failed.

#### 40.6.6.2.5 status\_t HAL\_CODEC\_SGTL5000\_SetMute ( void \* *handle*, uint32\_t *playChannel*, bool *isMute* )

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

Returns

kStatus\_Success is success, else configure failed.

#### 40.6.6.2.6 status\_t HAL\_CODEC\_SGTL5000\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* )

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.6.6.2.7 status\_t HAL\_CODEC\_SGTL5000\_SetRecord ( void \* *handle*, uint32\_t *recordSource* )

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.6.6.2.8 status\_t HAL\_CODEC\_SGTL5000\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* )

Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

Returns

kStatus\_Success is success, else configure failed.

#### 40.6.6.2.9 status\_t HAL\_CODEC\_SGTL5000\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.6.6.2.10 status\_t HAL\_CODEC\_SGTL5000\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* )

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.6.6.2.11 static status\_t HAL\_CODEC\_Init ( void \* *handle*, void \* *config* ) [inline], [static]

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

Returns

kStatus\_Success is success, else initial failed.

#### 40.6.6.2.12 static status\_t HAL\_CODEC\_Deinit ( void \* *handle* ) [inline], [static]

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

#### 40.6.6.2.13 static status\_t HAL\_CODEC\_SetFormat( void \* *handle*, uint32\_t *mclk*, uint32\_t *sampleRate*, uint32\_t *bitWidth* ) [inline], [static]

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

Returns

kStatus\_Success is success, else configure failed.

#### 40.6.6.2.14 static status\_t HAL\_CODEC\_SetVolume( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

Returns

kStatus\_Success is success, else configure failed.

#### 40.6.6.2.15 static status\_t HAL\_CODEC\_SetMute( void \* *handle*, uint32\_t *playChannel*, bool *isMute* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

Returns

kStatus\_Success is success, else configure failed.

#### 40.6.6.2.16 static status\_t HAL\_CODEC\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* ) [inline], [static]

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.6.6.2.17 static status\_t HAL\_CODEC\_SetRecord ( void \* *handle*, uint32\_t *recordSource* ) [inline], [static]

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.6.6.2.18 static status\_t HAL\_CODEC\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* ) [inline], [static]

Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |

Returns

kStatus\_Success is success, else configure failed.

**40.6.6.2.19 static status\_t HAL\_CODEC\_SetPlay ( void \* *handle*, uint32\_t *playSource* ) [inline], [static]**

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

**40.6.6.2.20 static status\_t HAL\_CODEC\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* ) [inline], [static]**

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

## 40.7 WM8960 Driver

### 40.7.1 Overview

The wm8960 driver provides a codec control interface.

## Data Structures

- struct `wm8960_audio_format_t`  
*wm8960 audio format More...*
- struct `wm8960_master_sysclk_config_t`  
*wm8960 master system clock configuration More...*
- struct `wm8960_config_t`  
*Initialize structure of WM8960. More...*
- struct `wm8960_handle_t`  
*wm8960 codec handler More...*

## Macros

- #define `WM8960_I2C_HANDLER_SIZE` CODEC\_I2C\_MASTER\_HANDLER\_SIZE  
*wm8960 handle size*
- #define `WM8960_LINVOL` 0x0U  
*Define the register address of WM8960.*
- #define `WM8960_CACHEREGNUM` 56U  
*Cache register number.*
- #define `WM8960_CLOCK2_BCLK_DIV_MASK` 0xFU  
*WM8960 CLOCK2 bits.*
- #define `WM8960_IFACE1_FORMAT_MASK` 0x03U  
*WM8960\_IFACE1 FORMAT bits.*
- #define `WM8960_IFACE1_WL_MASK` 0x0CU  
*WM8960\_IFACE1 WL bits.*
- #define `WM8960_IFACE1_LRP_MASK` 0x10U  
*WM8960\_IFACE1 LRP bit.*
- #define `WM8960_IFACE1_DLRSWAP_MASK` 0x20U  
*WM8960\_IFACE1 DLRSWAP bit.*
- #define `WM8960_IFACE1_MS_MASK` 0x40U  
*WM8960\_IFACE1 MS bit.*
- #define `WM8960_IFACE1_BCLKINV_MASK` 0x80U  
*WM8960\_IFACE1 BCLKINV bit.*
- #define `WM8960_IFACE1_ALRSWAP_MASK` 0x100U  
*WM8960\_IFACE1 ALRSWAP bit.*
- #define `WM8960_POWER1_VREF_MASK` 0x40U  
*WM8960\_POWER1.*
- #define `WM8960_POWER2_DACL_MASK` 0x100U  
*WM8960\_POWER2.*
- #define `WM8960_I2C_ADDR` 0x1A  
*WM8960 I2C address.*
- #define `WM8960_I2C_BAUDRATE` (100000U)

- WM8960 I<sub>2</sub>C baudrate.  
• #define WM8960\_ADC\_MAX\_VOLUME\_vVALUE 0xFFU  
WM8960 maximum volume value.

## Enumerations

- enum wm8960\_module\_t {
 kWM8960\_ModuleADC = 0,  
 kWM8960\_ModuleDAC = 1,  
 kWM8960\_ModuleVREF = 2,  
 kWM8960\_ModuleHP = 3,  
 kWM8960\_ModuleMICB = 4,  
 kWM8960\_ModuleMIC = 5,  
 kWM8960\_ModuleLineIn = 6,  
 kWM8960\_ModuleLineOut = 7,  
 kWM8960\_ModuleSpeaker = 8,  
 kWM8960\_ModuleOMIX = 9
 }  
*Modules in WM8960 board.*
- enum {
 kWM8960\_HeadphoneLeft = 1,  
 kWM8960\_HeadphoneRight = 2,  
 kWM8960\_SpeakerLeft = 4,  
 kWM8960\_SpeakerRight = 8
 }  
*wm8960 play channel*
- enum wm8960\_play\_source\_t {
 kWM8960\_PlaySourcePGA = 1,  
 kWM8960\_PlaySourceInput = 2,  
 kWM8960\_PlaySourceDAC = 4
 }  
*wm8960 play source*
- enum wm8960\_route\_t {
 kWM8960\_RouteBypass = 0,  
 kWM8960\_RoutePlayback = 1,  
 kWM8960\_RoutePlaybackandRecord = 2,  
 kWM8960\_RouteRecord = 5
 }  
*WM8960 data route.*
- enum wm8960\_protocol\_t {
 kWM8960\_BusI2S = 2,  
 kWM8960\_BusLeftJustified = 1,  
 kWM8960\_BusRightJustified = 0,  
 kWM8960\_BusPCMA = 3,  
 kWM8960\_BusPCMB = 3 | (1 << 4)
 }  
*The audio data transfer protocol choice.*
- enum wm8960\_input\_t {

```

kWM8960_InputClosed = 0,
kWM8960_InputSingleEndedMic = 1,
kWM8960_InputDifferentialMicInput2 = 2,
kWM8960_InputDifferentialMicInput3 = 3,
kWM8960_InputLineINPUT2 = 4,
kWM8960_InputLineINPUT3 = 5 }

 wm8960 input source
• enum {
 kWM8960_AudioSampleRate8KHz = 8000U,
 kWM8960_AudioSampleRate11025Hz = 11025U,
 kWM8960_AudioSampleRate12KHz = 12000U,
 kWM8960_AudioSampleRate16KHz = 16000U,
 kWM8960_AudioSampleRate22050Hz = 22050U,
 kWM8960_AudioSampleRate24KHz = 24000U,
 kWM8960_AudioSampleRate32KHz = 32000U,
 kWM8960_AudioSampleRate44100Hz = 44100U,
 kWM8960_AudioSampleRate48KHz = 48000U,
 kWM8960_AudioSampleRate96KHz = 96000U,
 kWM8960_AudioSampleRate192KHz = 192000U,
 kWM8960_AudioSampleRate384KHz = 384000U }

 audio sample rate definition
• enum {
 kWM8960_AudioBitWidth16bit = 16U,
 kWM8960_AudioBitWidth20bit = 20U,
 kWM8960_AudioBitWidth24bit = 24U,
 kWM8960_AudioBitWidth32bit = 32U }

 audio bit width
• enum wm8960_sysclk_source_t {
 kWM8960_SysClkSourceMclk = 0U,
 kWM8960_SysClkSourceInternalPLL = 1U }

 wm8960 sysclk source

```

## Functions

- **status\_t WM8960\_Init** (**wm8960\_handle\_t** \*handle, const **wm8960\_config\_t** \*config)  
*WM8960 initialize function.*
- **status\_t WM8960\_Deinit** (**wm8960\_handle\_t** \*handle)  
*Deinit the WM8960 codec.*
- **status\_t WM8960\_SetDataRoute** (**wm8960\_handle\_t** \*handle, **wm8960\_route\_t** route)  
*Set audio data route in WM8960.*
- **status\_t WM8960\_SetLeftInput** (**wm8960\_handle\_t** \*handle, **wm8960\_input\_t** input)  
*Set left audio input source in WM8960.*
- **status\_t WM8960\_SetRightInput** (**wm8960\_handle\_t** \*handle, **wm8960\_input\_t** input)  
*Set right audio input source in WM8960.*
- **status\_t WM8960\_SetProtocol** (**wm8960\_handle\_t** \*handle, **wm8960\_protocol\_t** protocol)  
*Set the audio transfer protocol.*

- void [WM8960\\_SetMasterSlave](#) (wm8960\_handle\_t \*handle, bool master)  
*Set WM8960 as master or slave.*
- status\_t [WM8960\\_SetVolume](#) (wm8960\_handle\_t \*handle, wm8960\_module\_t module, uint32\_t volume)  
*Set the volume of different modules in WM8960.*
- uint32\_t [WM8960\\_GetVolume](#) (wm8960\_handle\_t \*handle, wm8960\_module\_t module)  
*Get the volume of different modules in WM8960.*
- status\_t [WM8960\\_SetMute](#) (wm8960\_handle\_t \*handle, wm8960\_module\_t module, bool isEnabled)  
*Mute modules in WM8960.*
- status\_t [WM8960\\_SetModule](#) (wm8960\_handle\_t \*handle, wm8960\_module\_t module, bool isEnabled)  
*Enable/disable expected devices.*
- status\_t [WM8960\\_SetPlay](#) (wm8960\_handle\_t \*handle, uint32\_t playSource)  
*SET the WM8960 play source.*
- status\_t [WM8960\\_ConfigDataFormat](#) (wm8960\_handle\_t \*handle, uint32\_t sysclk, uint32\_t sample\_rate, uint32\_t bits)  
*Configure the data format of audio data.*
- status\_t [WM8960\\_SetJackDetect](#) (wm8960\_handle\_t \*handle, bool isEnabled)  
*Enable/disable jack detect feature.*
- status\_t [WM8960\\_WriteReg](#) (wm8960\_handle\_t \*handle, uint8\_t reg, uint16\_t val)  
*Write register to WM8960 using I2C.*
- status\_t [WM8960\\_ReadReg](#) (uint8\_t reg, uint16\_t \*val)  
*Read register from WM8960 using I2C.*
- status\_t [WM8960\\_ModifyReg](#) (wm8960\_handle\_t \*handle, uint8\_t reg, uint16\_t mask, uint16\_t val)  
*Modify some bits in the register using I2C.*

## Driver version

- #define [FSL\\_WM8960\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 2, 1))  
*CLOCK driver version 2.2.1.*

### 40.7.2 Data Structure Documentation

#### 40.7.2.1 struct [wm8960\\_audio\\_format\\_t](#)

##### Data Fields

- uint32\_t [mclk\\_HZ](#)  
*master clock frequency*
- uint32\_t [sampleRate](#)  
*sample rate*
- uint32\_t [bitWidth](#)  
*bit width*

#### 40.7.2.2 struct `wm8960_master_sysclk_config_t`

##### Data Fields

- `wm8960_sysclk_source_t sysclkSource`  
*sysclk source*
- `uint32_t sysclkFreq`  
*PLL output frequency value.*

#### 40.7.2.3 struct `wm8960_config_t`

##### Data Fields

- `wm8960_route_t route`  
*Audio data route.*
- `wm8960_protocol_t bus`  
*Audio transfer protocol.*
- `wm8960_audio_format_t format`  
*Audio format.*
- `bool master_slave`  
*Master or slave.*
- `wm8960_master_sysclk_config_t masterClock`  
*master clock configurations*
- `bool enableSpeaker`  
*True means enable class D speaker as output, false means no.*
- `wm8960_input_t leftInputSource`  
*Left input source for WM8960.*
- `wm8960_input_t rightInputSource`  
*Right input source for WM8960.*
- `wm8960_play_source_t playSource`  
*play source*
- `uint8_t slaveAddress`  
*wm8960 device address*
- `codec_i2c_config_t i2cConfig`  
*i2c configuration*

##### Field Documentation

(1) `wm8960_route_t wm8960_config_t::route`

(2) `bool wm8960_config_t::master_slave`

#### 40.7.2.4 struct `wm8960_handle_t`

##### Data Fields

- `const wm8960_config_t * config`  
*wm8904 config pointer*
- `uint8_t i2cHandle [WM8960_I2C_HANDLER_SIZE]`  
*i2c handle*

### 40.7.3 Macro Definition Documentation

#### 40.7.3.1 #define WM8960\_LINVOL 0x0U

#### 40.7.3.2 #define WM8960\_I2C\_ADDR 0x1A

### 40.7.4 Enumeration Type Documentation

#### 40.7.4.1 enum wm8960\_module\_t

Enumerator

*kWM8960\_ModuleADC* ADC module in WM8960.  
*kWM8960\_ModuleDAC* DAC module in WM8960.  
*kWM8960\_ModuleVREF* VREF module.  
*kWM8960\_ModuleHP* Headphone.  
*kWM8960\_ModuleMICB* Mic bias.  
*kWM8960\_ModuleMIC* Input Mic.  
*kWM8960\_ModuleLineIn* Analog in PGA.  
*kWM8960\_ModuleLineOut* Line out module.  
*kWM8960\_ModuleSpeaker* Speaker module.  
*kWM8960\_ModuleOMIX* Output mixer.

#### 40.7.4.2 anonymous enum

Enumerator

*kWM8960\_HeadphoneLeft* wm8960 headphone left channel  
*kWM8960\_HeadphoneRight* wm8960 headphone right channel  
*kWM8960\_SpeakerLeft* wm8960 speaker left channel  
*kWM8960\_SpeakerRight* wm8960 speaker right channel

#### 40.7.4.3 enum wm8960\_play\_source\_t

Enumerator

*kWM8960\_PlaySourcePGA* wm8960 play source PGA  
*kWM8960\_PlaySourceInput* wm8960 play source Input  
*kWM8960\_PlaySourceDAC* wm8960 play source DAC

#### 40.7.4.4 enum wm8960\_route\_t

Only provide some typical data route, not all route listed. Note: Users cannot combine any routes, once a new route is set, the previous one would be replaced.

Enumerator

*kWM8960\_RouteBypass* LINEIN->Headphone.  
*kWM8960\_RoutePlayback* I2SIN->DAC->Headphone.  
*kWM8960\_RoutePlaybackandRecord* I2SIN->DAC->Headphone, LINEIN->ADC->I2SOUT.  
*kWM8960\_RouteRecord* LINEIN->ADC->I2SOUT.

#### 40.7.4.5 enum **wm8960\_protocol\_t**

WM8960 only supports I2S format and PCM format.

Enumerator

*kWM8960\_BusI2S* I2S type.  
*kWM8960\_BusLeftJustified* Left justified mode.  
*kWM8960\_BusRightJustified* Right justified mode.  
*kWM8960\_BusPCMA* PCM A mode.  
*kWM8960\_BusPCMB* PCM B mode.

#### 40.7.4.6 enum **wm8960\_input\_t**

Enumerator

*kWM8960\_InputClosed* Input device is closed.  
*kWM8960\_InputSingleEndedMic* Input as single ended mic, only use L/RINPUT1.  
*kWM8960\_InputDifferentialMicInput2* Input as differential mic, use L/RINPUT1 and L/RINPUT2.  
*kWM8960\_InputDifferentialMicInput3* Input as differential mic, use L/RINPUT1 and L/RINPUT3.  
*kWM8960\_InputLineINPUT2* Input as line input, only use L/RINPUT2.  
*kWM8960\_InputLineINPUT3* Input as line input, only use L/RINPUT3.

#### 40.7.4.7 anonymous enum

Enumerator

*kWM8960\_AudioSampleRate8KHz* Sample rate 8000 Hz.  
*kWM8960\_AudioSampleRate11025Hz* Sample rate 11025 Hz.  
*kWM8960\_AudioSampleRate12KHz* Sample rate 12000 Hz.  
*kWM8960\_AudioSampleRate16KHz* Sample rate 16000 Hz.  
*kWM8960\_AudioSampleRate22050Hz* Sample rate 22050 Hz.  
*kWM8960\_AudioSampleRate24KHz* Sample rate 24000 Hz.  
*kWM8960\_AudioSampleRate32KHz* Sample rate 32000 Hz.  
*kWM8960\_AudioSampleRate44100Hz* Sample rate 44100 Hz.  
*kWM8960\_AudioSampleRate48KHz* Sample rate 48000 Hz.

*kWM8960\_AudioSampleRate96KHz* Sample rate 96000 Hz.  
*kWM8960\_AudioSampleRate192KHz* Sample rate 192000 Hz.  
*kWM8960\_AudioSampleRate384KHz* Sample rate 384000 Hz.

#### 40.7.4.8 anonymous enum

Enumerator

*kWM8960\_AudioBitWidth16bit* audio bit width 16  
*kWM8960\_AudioBitWidth20bit* audio bit width 20  
*kWM8960\_AudioBitWidth24bit* audio bit width 24  
*kWM8960\_AudioBitWidth32bit* audio bit width 32

#### 40.7.4.9 enum **wm8960\_sysclk\_source\_t**

Enumerator

*kWM8960\_SysClkSourceMclk* sysclk source from external MCLK  
*kWM8960\_SysClkSourceInternalPLL* sysclk source from internal PLL

### 40.7.5 Function Documentation

#### 40.7.5.1 status\_t **WM8960\_Init** ( **wm8960\_handle\_t \* handle**, **const wm8960\_config\_t \* config** )

The second parameter is NULL to WM8960 in this version. If users want to change the settings, they have to use `wm8960_write_reg()` or `wm8960_modify_reg()` to set the register value of WM8960. Note: If the `codec_config` is NULL, it would initialize WM8960 using default settings. The default setting: `codec_config->route = kWM8960_RoutePlaybackandRecord` `codec_config->bus = kWM8960_BusI2S` `codec_config->master = slave`

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>handle</i> | WM8960 handle structure.        |
| <i>config</i> | WM8960 configuration structure. |

#### 40.7.5.2 status\_t **WM8960\_Deinit** ( **wm8960\_handle\_t \* handle** )

This function close all modules in WM8960 to save power.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>handle</i> | WM8960 handle structure pointer. |
|---------------|----------------------------------|

#### 40.7.5.3 status\_t WM8960\_SetDataRoute ( *wm8960\_handle\_t \* handle, wm8960\_route\_t route* )

This function would set the data route according to route. The route cannot be combined, as all route would enable different modules. Note: If a new route is set, the previous route would not work.

Parameters

|               |                             |
|---------------|-----------------------------|
| <i>handle</i> | WM8960 handle structure.    |
| <i>route</i>  | Audio data route in WM8960. |

#### 40.7.5.4 status\_t WM8960\_SetLeftInput ( *wm8960\_handle\_t \* handle, wm8960\_input\_t input* )

Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8960 handle structure. |
| <i>input</i>  | Audio input source.      |

#### 40.7.5.5 status\_t WM8960\_SetRightInput ( *wm8960\_handle\_t \* handle, wm8960\_input\_t input* )

Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8960 handle structure. |
| <i>input</i>  | Audio input source.      |

#### 40.7.5.6 status\_t WM8960\_SetProtocol ( *wm8960\_handle\_t \* handle, wm8960\_protocol\_t protocol* )

WM8960 only supports I2S, left justified, right justified, PCM A, PCM B format.

Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>handle</i>   | WM8960 handle structure.      |
| <i>protocol</i> | Audio data transfer protocol. |

#### 40.7.5.7 void WM8960\_SetMasterSlave ( **wm8960\_handle\_t \* handle, bool master** )

Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>handle</i> | WM8960 handle structure.               |
| <i>master</i> | 1 represent master, 0 represent slave. |

#### 40.7.5.8 status\_t WM8960\_SetVolume ( **wm8960\_handle\_t \* handle, wm8960\_module\_t module, uint32\_t volume** )

This function would set the volume of WM8960 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Module:kWM8960\_ModuleADC, volume range value: 0 is mute, 1-255 is -97db to 30db  
 Module:kWM8960\_ModuleDAC, volume range value: 0 is mute, 1-255 is -127db to 0db  
 Module:kWM8960\_ModuleHP, volume range value: 0 - 2F is mute, 0x30 - 0x7F is -73db to 6db  
 Module:kWM8960\_ModuleLineIn, volume range value: 0 - 0x3F is -17.25db to 30db  
 Module:kWM8960\_ModuleSpeaker, volume range value: 0 - 2F is mute, 0x30 - 0x7F is -73db to 6db

Parameters

|               |                                                                |
|---------------|----------------------------------------------------------------|
| <i>handle</i> | WM8960 handle structure.                                       |
| <i>module</i> | Module to set volume, it can be ADC, DAC, Headphone and so on. |
| <i>volume</i> | Volume value need to be set.                                   |

#### 40.7.5.9 uint32\_t WM8960\_GetVolume ( **wm8960\_handle\_t \* handle, wm8960\_module\_t module** )

This function gets the volume of WM8960 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

|               |                                                                |
|---------------|----------------------------------------------------------------|
| <i>handle</i> | WM8960 handle structure.                                       |
| <i>module</i> | Module to set volume, it can be ADC, DAC, Headphone and so on. |

Returns

Volume value of the module.

#### 40.7.5.10 status\_t WM8960\_SetMute ( *wm8960\_handle\_t \* handle, wm8960\_module\_t module, bool isEnabled* )

Parameters

|                  |                                   |
|------------------|-----------------------------------|
| <i>handle</i>    | WM8960 handle structure.          |
| <i>module</i>    | Modules need to be mute.          |
| <i>isEnabled</i> | Mute or unmute, 1 represent mute. |

#### 40.7.5.11 status\_t WM8960\_SetModule ( *wm8960\_handle\_t \* handle, wm8960\_module\_t module, bool isEnabled* )

Parameters

|                  |                            |
|------------------|----------------------------|
| <i>handle</i>    | WM8960 handle structure.   |
| <i>module</i>    | Module expected to enable. |
| <i>isEnabled</i> | Enable or disable moudles. |

#### 40.7.5.12 status\_t WM8960\_SetPlay ( *wm8960\_handle\_t \* handle, uint32\_t playSource* )

Parameters

|                   |                                                                                                                                                                                                      |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>     | WM8960 handle structure.                                                                                                                                                                             |
| <i>playSource</i> | play source , can be a value combine of kWM8960_ModuleHeadphoneSourcePG-A, kWM8960_ModuleHeadphoneSourceDAC, kWM8960_ModulePlaySourceInput, kWM8960_ModulePlayMonoRight, kWM8960_ModulePlayMonoLeft. |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise..

#### 40.7.5.13 status\_t WM8960\_ConfigDataFormat ( *wm8960\_handle\_t \* handle, uint32\_t sysclk, uint32\_t sample\_rate, uint32\_t bits* )

This function would configure the registers about the sample rate, bit depths.

Parameters

|                    |                                                                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>      | WM8960 handle structure pointer.                                                                                                          |
| <i>sysclk</i>      | system clock of the codec which can be generated by MCLK or PLL output.                                                                   |
| <i>sample_rate</i> | Sample rate of audio file running in WM8960. WM8960 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate. |
| <i>bits</i>        | Bit depth of audio file (WM8960 only supports 16bit, 20bit, 24bit and 32 bit in HW).                                                      |

**40.7.5.14 status\_t WM8960\_SetJackDetect ( *wm8960\_handle\_t \* handle, bool isEnabled* )**

Parameters

|                  |                            |
|------------------|----------------------------|
| <i>handle</i>    | WM8960 handle structure.   |
| <i>isEnabled</i> | Enable or disable moudles. |

**40.7.5.15 status\_t WM8960\_WriteReg ( *wm8960\_handle\_t \* handle, uint8\_t reg, uint16\_t val* )**

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>handle</i> | WM8960 handle structure.                |
| <i>reg</i>    | The register address in WM8960.         |
| <i>val</i>    | Value needs to write into the register. |

**40.7.5.16 status\_t WM8960\_ReadReg ( *uint8\_t reg, uint16\_t \* val* )**

Parameters

|            |                                 |
|------------|---------------------------------|
| <i>reg</i> | The register address in WM8960. |
| <i>val</i> | Value written to.               |

**40.7.5.17 status\_t WM8960\_ModifyReg ( *wm8960\_handle\_t \* handle, uint8\_t reg, uint16\_t mask, uint16\_t val* )**

## Parameters

|               |                                                                                  |
|---------------|----------------------------------------------------------------------------------|
| <i>handle</i> | WM8960 handle structure.                                                         |
| <i>reg</i>    | The register address in WM8960.                                                  |
| <i>mask</i>   | The mask code for the bits want to write. The bit you want to write should be 0. |
| <i>val</i>    | Value needs to write into the register.                                          |

## 40.7.6 WM8960 Adapter

### 40.7.6.1 Overview

The wm8960 adapter provides a codec unify control interface.

#### Macros

- #define `HAL_CODEC_WM8960_HANDLER_SIZE` (`WM8960_I2C_HANDLER_SIZE + 4`)  
*codec handler size*

#### Functions

- `status_t HAL_CODEC_WM8960_Init` (void \*handle, void \*config)  
*Codec initialization.*
- `status_t HAL_CODEC_WM8960_Deinit` (void \*handle)  
*Codec de-initilization.*
- `status_t HAL_CODEC_WM8960_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- `status_t HAL_CODEC_WM8960_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- `status_t HAL_CODEC_WM8960_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- `status_t HAL_CODEC_WM8960_SetPower` (void \*handle, uint32\_t module, bool powerOn)  
*set audio codec module power.*
- `status_t HAL_CODEC_WM8960_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- `status_t HAL_CODEC_WM8960_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- `status_t HAL_CODEC_WM8960_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- `status_t HAL_CODEC_WM8960_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*
- static `status_t HAL_CODEC_Init` (void \*handle, void \*config)  
*Codec initilization.*
- static `status_t HAL_CODEC_Deinit` (void \*handle)  
*Codec de-initilization.*
- static `status_t HAL_CODEC_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- static `status_t HAL_CODEC_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- static `status_t HAL_CODEC_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- static `status_t HAL_CODEC_SetPower` (void \*handle, uint32\_t module, bool powerOn)

- static `status_t HAL_CODEC_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- static `status_t HAL_CODEC_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- static `status_t HAL_CODEC_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- static `status_t HAL_CODEC_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

#### 40.7.6.2 Function Documentation

##### 40.7.6.2.1 `status_t HAL_CODEC_WM8960_Init( void * handle, void * config )`

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

Returns

kStatus\_Success is success, else initial failed.

##### 40.7.6.2.2 `status_t HAL_CODEC_WM8960_Deinit( void * handle )`

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

##### 40.7.6.2.3 `status_t HAL_CODEC_WM8960_SetFormat( void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth )`

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

Returns

kStatus\_Success is success, else configure failed.

#### 40.7.6.2.4 status\_t HAL\_CODEC\_WM8960\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

Returns

kStatus\_Success is success, else configure failed.

#### 40.7.6.2.5 status\_t HAL\_CODEC\_WM8960\_SetMute ( void \* *handle*, uint32\_t *playChannel*, bool *isMute* )

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

Returns

kStatus\_Success is success, else configure failed.

#### 40.7.6.2.6 status\_t HAL\_CODEC\_WM8960\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* )

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.7.6.2.7 status\_t HAL\_CODEC\_WM8960\_SetRecord ( void \* *handle*, uint32\_t *recordSource* )

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.7.6.2.8 status\_t HAL\_CODEC\_WM8960\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* )

Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

Returns

kStatus\_Success is success, else configure failed.

#### 40.7.6.2.9 status\_t HAL\_CODEC\_WM8960\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.7.6.2.10 status\_t HAL\_CODEC\_WM8960\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* )

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.7.6.2.11 static status\_t HAL\_CODEC\_Init ( void \* *handle*, void \* *config* ) [inline], [static]

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

Returns

kStatus\_Success is success, else initial failed.

#### 40.7.6.2.12 static status\_t HAL\_CODEC\_Deinit ( void \* *handle* ) [inline], [static]

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

#### 40.7.6.2.13 static status\_t HAL\_CODEC\_SetFormat( void \* *handle*, uint32\_t *mclk*, uint32\_t *sampleRate*, uint32\_t *bitWidth* ) [inline], [static]

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

Returns

kStatus\_Success is success, else configure failed.

#### 40.7.6.2.14 static status\_t HAL\_CODEC\_SetVolume( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

Returns

kStatus\_Success is success, else configure failed.

#### 40.7.6.2.15 static status\_t HAL\_CODEC\_SetMute( void \* *handle*, uint32\_t *playChannel*, bool *isMute* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

Returns

kStatus\_Success is success, else configure failed.

#### 40.7.6.2.16 static status\_t HAL\_CODEC\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* ) [inline], [static]

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.7.6.2.17 static status\_t HAL\_CODEC\_SetRecord ( void \* *handle*, uint32\_t *recordSource* ) [inline], [static]

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 40.7.6.2.18 static status\_t HAL\_CODEC\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* ) [inline], [static]

Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

Returns

kStatus\_Success is success, else configure failed.

**40.7.6.2.19 static status\_t HAL\_CODEC\_SetPlay ( void \* *handle*, uint32\_t *playSource* ) [inline], [static]**

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

**40.7.6.2.20 static status\_t HAL\_CODEC\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* ) [inline], [static]**

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

## 40.8 WM8904 Driver

### 40.8.1 Overview

The wm8904 driver provides a codec control interface.

### Data Structures

- struct [wm8904\\_fll\\_config\\_t](#)  
*wm8904 fll configuration* [More...](#)
- struct [wm8904\\_audio\\_format\\_t](#)  
*Audio format configuration.* [More...](#)
- struct [wm8904\\_config\\_t](#)  
*Configuration structure of WM8904.* [More...](#)
- struct [wm8904\\_handle\\_t](#)  
*wm8904 codec handler* [More...](#)

### Macros

- #define [WM8904\\_I2C\\_HANDLER\\_SIZE](#) (CODEC\_I2C\_MASTER\_HANDLER\_SIZE)  
*wm8904 handle size*
- #define [WM8904\\_DEBUG\\_REGISTER](#) 0  
*wm8904 debug macro*
- #define [WM8904\\_RESET](#) (0x00)  
*WM8904 register map.*
- #define [WM8904\\_I2C\\_ADDRESS](#) (0x1A)  
*WM8904 I2C address.*
- #define [WM8904\\_I2C\\_BITRATE](#) (400000U)  
*WM8904 I2C bit rate.*
- #define [WM8904\\_MAP\\_HEADPHONE\\_LINEOUT\\_MAX\\_VOLUME](#) 0x3FU  
*WM8904 maximum volume.*

### Enumerations

- enum {
   
*kStatus\_WM8904\_Success* = 0x0,  
*kStatus\_WM8904\_Fail* = 0x1
 }
   
*WM8904 status return codes.*
- enum {
   
*kWM8904\_LRCPolarityNormal* = 0U,  
*kWM8904\_LRCPolarityInverted* = 1U << 4U
 }
   
*WM8904 lrc polarity.*
- enum [wm8904\\_module\\_t](#) {

- ```
kWM8904_ModuleADC = 0,
kWM8904_ModuleDAC = 1,
kWM8904_ModulePGA = 2,
kWM8904_ModuleHeadphone = 3,
kWM8904_ModuleLineout = 4 }
```

wm8904 module value
- enum

```
wm8904 play channel
```
- enum `wm8904_timeslot_t` {

```
kWM8904_TimeSlot0 = 0U,
kWM8904_TimeSlot1 = 1U }
```

WM8904 time slot.
- enum `wm8904_protocol_t` {

```
kWM8904_ProtocolI2S = 0x2,
kWM8904_ProtocolLeftJustified = 0x1,
kWM8904_ProtocolRightJustified = 0x0,
kWM8904_ProtocolPCMA = 0x3,
kWM8904_ProtocolPCMB = 0x3 | (1 << 4) }
```

The audio data transfer protocol.
- enum `wm8904_fs_ratio_t` {

```
kWM8904_FsRatio64X = 0x0,
kWM8904_FsRatio128X = 0x1,
kWM8904_FsRatio192X = 0x2,
kWM8904_FsRatio256X = 0x3,
kWM8904_FsRatio384X = 0x4,
kWM8904_FsRatio512X = 0x5,
kWM8904_FsRatio768X = 0x6,
kWM8904_FsRatio1024X = 0x7,
kWM8904_FsRatio1408X = 0x8,
kWM8904_FsRatio1536X = 0x9 }
```

The SYSCLK / fs ratio.
- enum `wm8904_sample_rate_t` {

```
kWM8904_SampleRate8kHz = 0x0,
kWM8904_SampleRate12kHz = 0x1,
kWM8904_SampleRate16kHz = 0x2,
kWM8904_SampleRate24kHz = 0x3,
kWM8904_SampleRate32kHz = 0x4,
kWM8904_SampleRate48kHz = 0x5,
kWM8904_SampleRate11025Hz = 0x6,
kWM8904_SampleRate22050Hz = 0x7,
kWM8904_SampleRate44100Hz = 0x8 }
```

Sample rate.
- enum `wm8904_bit_width_t` {

```
kWM8904_BitWidth16 = 0x0,
kWM8904_BitWidth20 = 0x1,
kWM8904_BitWidth24 = 0x2,
```

```

kWM8904_BitWidth32 = 0x3 }

Bit width.
• enum {
    kWM8904_RecordSourceDifferentialLine = 1U,
    kWM8904_RecordSourceLineInput = 2U,
    kWM8904_RecordSourceDifferentialMic = 4U,
    kWM8904_RecordSourceDigitalMic = 8U }
    wm8904 record source
• enum {
    kWM8904_RecordChannelLeft1 = 1U,
    kWM8904_RecordChannelLeft2 = 2U,
    kWM8904_RecordChannelLeft3 = 4U,
    kWM8904_RecordChannelRight1 = 1U,
    kWM8904_RecordChannelRight2 = 2U,
    kWM8904_RecordChannelRight3 = 4U,
    kWM8904_RecordChannelDifferentialPositive1 = 1U,
    kWM8904_RecordChannelDifferentialPositive2 = 2U,
    kWM8904_RecordChannelDifferentialPositive3 = 4U,
    kWM8904_RecordChannelDifferentialNegative1 = 8U,
    kWM8904_RecordChannelDifferentialNegative2 = 16U,
    kWM8904_RecordChannelDifferentialNegative3 = 32U }
    wm8904 record channel
• enum {
    kWM8904_PlaySourcePGA = 1U,
    kWM8904_PlaySourceDAC = 4U }
    wm8904 play source
• enum wm8904\_sys\_clk\_source\_t {
    kWM8904_SysClkSourceMCLK = 0U,
    kWM8904_SysClkSourceFLL = 1U << 14 }
    wm8904 system clock source
• enum wm8904\_fll\_clk\_source\_t { kWM8904_FLLClkSourceMCLK = 0U }
    wm8904 fll clock source

```

Functions

- [status_t WM8904_WriteRegister \(wm8904_handle_t *handle, uint8_t reg, uint16_t value\)](#)
WM8904 write register.
- [status_t WM8904_ReadRegister \(wm8904_handle_t *handle, uint8_t reg, uint16_t *value\)](#)
WM8904 write register.
- [status_t WM8904_ModifyRegister \(wm8904_handle_t *handle, uint8_t reg, uint16_t mask, uint16_t value\)](#)
WM8904 modify register.
- [status_t WM8904_Init \(wm8904_handle_t *handle, \[wm8904_config_t\]\(#\) *wm8904Config\)](#)
Initializes WM8904.
- [status_t WM8904_Deinit \(wm8904_handle_t *handle\)](#)
Deinitializes the WM8904 codec.
- [void WM8904_GetDefaultConfig \(\[wm8904_config_t\]\(#\) *config\)](#)

Fills the configuration structure with default values.

- **status_t WM8904_SetMasterSlave** (`wm8904_handle_t *handle, bool master`)

Sets WM8904 as master or slave.
- **status_t WM8904_SetMasterClock** (`wm8904_handle_t *handle, uint32_t sysclk, uint32_t sampleRate, uint32_t bitWidth`)

Sets WM8904 master clock configuration.
- **status_t WM8904_SetFLLConfig** (`wm8904_handle_t *handle, wm8904_fll_config_t *config`)

WM8904 set PLL configuration This function will enable the GPIO1 FLL clock output function, so user can see the generated fll output clock frequency from WM8904 GPIO1.
- **status_t WM8904_SetProtocol** (`wm8904_handle_t *handle, wm8904_protocol_t protocol`)

Sets the audio data transfer protocol.
- **status_t WM8904_SetAudioFormat** (`wm8904_handle_t *handle, uint32_t sysclk, uint32_t sampleRate, uint32_t bitWidth`)

Sets the audio data format.
- **status_t WM8904_CheckAudioFormat** (`wm8904_handle_t *handle, wm8904_audio_format_t *format, uint32_t mclkFreq`)

check and update the audio data format.
- **status_t WM8904_SetVolume** (`wm8904_handle_t *handle, uint16_t volumeLeft, uint16_t volumeRight`)

Sets the module output volume.
- **status_t WM8904_SetMute** (`wm8904_handle_t *handle, bool muteLeft, bool muteRight`)

Sets the headphone output mute.
- **status_t WM8904_SelectLRCPolarity** (`wm8904_handle_t *handle, uint32_t polarity`)

Select LRC polarity.
- **status_t WM8904_EnableDACTDMMMode** (`wm8904_handle_t *handle, wm8904_timeslot_t timeSlot`)

Enable WM8904 DAC time slot.
- **status_t WM8904_EnableADCTDMMMode** (`wm8904_handle_t *handle, wm8904_timeslot_t timeSlot`)

Enable WM8904 ADC time slot.
- **status_t WM8904_SetModulePower** (`wm8904_handle_t *handle, wm8904_module_t module, bool isEnabled`)

SET the module output power.
- **status_t WM8904_SetDACVolume** (`wm8904_handle_t *handle, uint8_t volume`)

SET the DAC module volume.
- **status_t WM8904_SetChannelVolume** (`wm8904_handle_t *handle, uint32_t channel, uint32_t volume`)

Sets the channel output volume.
- **status_t WM8904_SetRecord** (`wm8904_handle_t *handle, uint32_t recordSource`)

SET the WM8904 record source.
- **status_t WM8904_SetRecordChannel** (`wm8904_handle_t *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel`)

SET the WM8904 record source.
- **status_t WM8904_SetPlay** (`wm8904_handle_t *handle, uint32_t playSource`)

SET the WM8904 play source.
- **status_t WM8904_SetChannelMute** (`wm8904_handle_t *handle, uint32_t channel, bool isMute`)

Sets the channel mute.

Driver version

- #define FSL_WM8904_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))
WM8904 driver version 2.5.1.

40.8.2 Data Structure Documentation

40.8.2.1 struct wm8904_fll_config_t

Data Fields

- `wm8904_fll_clk_source_t source`
fll reference clock source
- `uint32_t refClock_HZ`
fll reference clock frequency
- `uint32_t outputClock_HZ`
fll output clock frequency

40.8.2.2 struct wm8904_audio_format_t

Data Fields

- `wm8904_fs_ratio_t fsRatio`
SYSCLK / fs ratio.
- `wm8904_sample_rate_t sampleRate`
Sample rate.
- `wm8904_bit_width_t bitWidth`
Bit width.

40.8.2.3 struct wm8904_config_t

Data Fields

- `bool master`
Master or slave.
- `wm8904_sys_clk_source_t sysClkSource`
system clock source
- `wm8904_fll_config_t * fll`
fll configuration
- `wm8904_protocol_t protocol`
Audio transfer protocol.
- `wm8904_audio_format_t format`
Audio format.
- `uint32_t mclk_HZ`
MCLK frequency value.
- `uint16_t recordSource`
record source

- `uint16_t recordChannelLeft`
record channel
- `uint16_t recordChannelRight`
record channel
- `uint16_t playSource`
play source
- `uint8_t slaveAddress`
code device slave address
- `codec_i2c_config_t i2cConfig`
i2c bus configuration

40.8.2.4 struct `wm8904_handle_t`

Data Fields

- `wm8904_config_t * config`
wm8904 config pointer
- `uint8_t i2cHandle [WM8904_I2C_HANDLER_SIZE]`
i2c handle

40.8.3 Macro Definition Documentation

40.8.3.1 `#define FSL_WM8904_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))`

40.8.3.2 `#define WM8904_I2C_ADDRESS (0x1A)`

40.8.3.3 `#define WM8904_I2C_BITRATE (400000U)`

40.8.4 Enumeration Type Documentation

40.8.4.1 anonymous enum

Enumerator

`kStatus_WM8904_Success` Success.
`kStatus_WM8904_Fail` Failure.

40.8.4.2 anonymous enum

Enumerator

`kWM8904_LRCPolarityNormal` LRC polarity normal.
`kWM8904_LRCPolarityInverted` LRC polarity inverted.

40.8.4.3 enum wm8904_module_t

Enumerator

- kWM8904_ModuleADC* module ADC
- kWM8904_ModuleDAC* module DAC
- kWM8904_ModulePGA* module PGA
- kWM8904_ModuleHeadphone* module headphone
- kWM8904_ModuleLineout* module line out

40.8.4.4 anonymous enum

40.8.4.5 enum wm8904_timeslot_t

Enumerator

- kWM8904_TimeSlot0* time slot0
- kWM8904_TimeSlot1* time slot1

40.8.4.6 enum wm8904_protocol_t

Enumerator

- kWM8904_ProtocolI2S* I2S type.
- kWM8904_ProtocolLeftJustified* Left justified mode.
- kWM8904_ProtocolRightJustified* Right justified mode.
- kWM8904_ProtocolPCMA* PCM A mode.
- kWM8904_ProtocolPCMB* PCM B mode.

40.8.4.7 enum wm8904_fs_ratio_t

Enumerator

- kWM8904_FsRatio64X* SYSCLK is $64 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio128X* SYSCLK is $128 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio192X* SYSCLK is $192 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio256X* SYSCLK is $256 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio384X* SYSCLK is $384 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio512X* SYSCLK is $512 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio768X* SYSCLK is $768 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio1024X* SYSCLK is $1024 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio1408X* SYSCLK is $1408 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio1536X* SYSCLK is $1536 * \text{sample rate} * \text{frame width}$.

40.8.4.8 enum `wm8904_sample_rate_t`

Enumerator

- kWM8904_SampleRate8kHz* 8 kHz
- kWM8904_SampleRate12kHz* 12kHz
- kWM8904_SampleRate16kHz* 16kHz
- kWM8904_SampleRate24kHz* 24kHz
- kWM8904_SampleRate32kHz* 32kHz
- kWM8904_SampleRate48kHz* 48kHz
- kWM8904_SampleRate11025Hz* 11.025kHz
- kWM8904_SampleRate22050Hz* 22.05kHz
- kWM8904_SampleRate44100Hz* 44.1kHz

40.8.4.9 enum `wm8904_bit_width_t`

Enumerator

- kWM8904_BitWidth16* 16 bits
- kWM8904_BitWidth20* 20 bits
- kWM8904_BitWidth24* 24 bits
- kWM8904_BitWidth32* 32 bits

40.8.4.10 anonymous enum

Enumerator

- kWM8904_RecordSourceDifferentialLine* record source from differential line
- kWM8904_RecordSourceLineInput* record source from line input
- kWM8904_RecordSourceDifferentialMic* record source from differential mic
- kWM8904_RecordSourceDigitalMic* record source from digital microphone

40.8.4.11 anonymous enum

Enumerator

- kWM8904_RecordChannelLeft1* left record channel 1
- kWM8904_RecordChannelLeft2* left record channel 2
- kWM8904_RecordChannelLeft3* left record channel 3
- kWM8904_RecordChannelRight1* right record channel 1
- kWM8904_RecordChannelRight2* right record channel 2
- kWM8904_RecordChannelRight3* right record channel 3
- kWM8904_RecordChannelDifferentialPositive1* differential positive record channel 1
- kWM8904_RecordChannelDifferentialPositive2* differential positive record channel 2
- kWM8904_RecordChannelDifferentialPositive3* differential positive record channel 3

- kWM8904_RecordChannelDifferentialNegative1* differential negative record channel 1
- kWM8904_RecordChannelDifferentialNegative2* differential negative record channel 2
- kWM8904_RecordChannelDifferentialNegative3* differential negative record channel 3

40.8.4.12 anonymous enum

Enumerator

- kWM8904_PlaySourcePGA* play source PGA, bypass ADC
- kWM8904_PlaySourceDAC* play source Input3

40.8.4.13 enum `wm8904_sys_clk_source_t`

Enumerator

- kWM8904_SysClkSourceMCLK* wm8904 system clock soure from MCLK
- kWM8904_SysClkSourceFLL* wm8904 system clock soure from FLL

40.8.4.14 enum `wm8904_fll_clk_source_t`

Enumerator

- kWM8904_FLLClkSourceMCLK* wm8904 FLL clock source from MCLK

40.8.5 Function Documentation

40.8.5.1 status_t `WM8904_WriteRegister` (`wm8904_handle_t * handle, uint8_t reg, uint16_t value`)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>reg</i>	register address.
<i>value</i>	value to write.

Returns

`kStatus_Success`, else failed.

40.8.5.2 status_t `WM8904_ReadRegister` (`wm8904_handle_t * handle, uint8_t reg, uint16_t * value`)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>reg</i>	register address.
<i>value</i>	value to read.

Returns

kStatus_Success, else failed.

40.8.5.3 status_t WM8904_ModifyRegister (*wm8904_handle_t * handle, uint8_t reg, uint16_t mask, uint16_t value*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>reg</i>	register address.
<i>mask</i>	register bits mask.
<i>value</i>	value to write.

Returns

kStatus_Success, else failed.

40.8.5.4 status_t WM8904_Init (*wm8904_handle_t * handle, wm8904_config_t * wm8904Config*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>wm8904Config</i>	WM8904 configuration structure.

40.8.5.5 status_t WM8904_Deinit (*wm8904_handle_t * handle*)

This function resets WM8904.

Parameters

<i>handle</i>	WM8904 handle structure.
---------------	--------------------------

Returns

kStatus_WM8904_Success if successful, different code otherwise.

40.8.5.6 void WM8904_GetDefaultConfig (*wm8904_config_t * config*)

The default values are:

```
master = false; protocol = kWM8904_ProtocolI2S; format.fsRatio = kWM8904_FsRatio64X; format.-sampleRate = kWM8904_SampleRate48kHz; format.bitWidth = kWM8904_BitWidth16;
```

Parameters

<i>config</i>	default configurations of wm8904.
---------------	-----------------------------------

40.8.5.7 status_t WM8904_SetMasterSlave (*wm8904_handle_t * handle, bool master*)

Deprecated DO NOT USE THIS API ANYMORE. IT HAS BEEN SUPERCEDED BY [WM8904_SetMasterClock](#)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>master</i>	true for master, false for slave.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

40.8.5.8 status_t WM8904_SetMasterClock (*wm8904_handle_t * handle, uint32_t sysclk, uint32_t sampleRate, uint32_t bitWidth*)

User should pay attention to the sysclk parameter ,When using external MCLK as system clock source, the value should be frequency of MCLK, when using FLL as system clock source, the value should be frequency of the output of FLL.

Parameters

<i>handle</i>	WM8904 handle structure.
<i>sysclk</i>	system clock source frequency.
<i>sampleRate</i>	sample rate
<i>bitWidth</i>	bit width

Returns

kStatus_WM8904_Success if successful, different code otherwise.

40.8.5.9 status_t WM8904_SetFLLConfig (*wm8904_handle_t * handle*, *wm8904_fll_config_t * config*)

Parameters

<i>handle</i>	wm8904 handler pointer.
<i>config</i>	FLL configuration pointer.

40.8.5.10 status_t WM8904_SetProtocol (*wm8904_handle_t * handle*, *wm8904_protocol_t protocol*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>protocol</i>	Audio transfer protocol.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

40.8.5.11 status_t WM8904_SetAudioFormat (*wm8904_handle_t * handle*, *uint32_t sysclk*, *uint32_t sampleRate*, *uint32_t bitWidth*)

User should pay attention to the sysclk parameter ,When using external MCLK as system clock source, the value should be frequency of MCLK, when using FLL as system clock source, the value should be frequency of the output of FLL.

Parameters

<i>handle</i>	WM8904 handle structure.
<i>sysclk</i>	system clock source frequency.
<i>sampleRate</i>	Sample rate frequency in Hz.
<i>bitWidth</i>	Audio data bit width.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

40.8.5.12 status_t WM8904_CheckAudioFormat (*wm8904_handle_t * handle*, *wm8904_audio_format_t * format*, *uint32_t mclkFreq*)

This api is used check the fsRatio setting based on the mclk and sample rate, if fsRatio setting is not correct, it will correct it according to mclk and sample rate.

Parameters

<i>handle</i>	WM8904 handle structure.
<i>format</i>	audio data format
<i>mclkFreq</i>	mclk frequency

Returns

kStatus_WM8904_Success if successful, different code otherwise.

40.8.5.13 status_t WM8904_SetVolume (*wm8904_handle_t * handle*, *uint16_t volumeLeft*, *uint16_t volumeRight*)

The parameter should be from 0 to 63. The resulting volume will be. 0 for -57DB, 63 for 6DB.

Parameters

<i>handle</i>	WM8904 handle structure.
---------------	--------------------------

<i>volumeLeft</i>	left channel volume.
<i>volumeRight</i>	right channel volume.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

40.8.5.14 status_t WM8904_SetMute (*wm8904_handle_t * handle, bool muteLeft, bool muteRight*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>muteLeft</i>	true to mute left channel, false to unmute.
<i>muteRight</i>	true to mute right channel, false to unmute.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

40.8.5.15 status_t WM8904_SelectLRCPolarity (*wm8904_handle_t * handle, uint32_t polarity*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>polarity</i>	LRC clock polarity.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

40.8.5.16 status_t WM8904_EnableDACTDMMode (*wm8904_handle_t * handle, wm8904_timeslot_t timeSlot*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>timeSlot</i>	timeslot number.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

40.8.5.17 status_t WM8904_EnableADCTDMMMode (*wm8904_handle_t * handle,* *wm8904_timeslot_t timeSlot*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>timeSlot</i>	timeslot number.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

40.8.5.18 status_t WM8904_SetModulePower (*wm8904_handle_t * handle,* *wm8904_module_t module, bool isEnabled*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>module</i>	wm8904 module.
<i>isEnabled</i> , <i>true</i>	is power on, false is power down.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

40.8.5.19 status_t WM8904_SetDACVolume (*wm8904_handle_t * handle, uint8_t volume* *)*

Parameters

<i>handle</i>	WM8904 handle structure.
<i>volume</i>	volume to be configured.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

40.8.5.20 status_t WM8904_SetChannelVolume (*wm8904_handle_t * handle, uint32_t channel, uint32_t volume*)

The parameter should be from 0 to 63. The resulting volume will be. 0 for -57dB, 63 for 6DB.

Parameters

<i>handle</i>	codec handle structure.
<i>channel</i>	codec channel.
<i>volume</i>	volume value from 0 -63.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

40.8.5.21 status_t WM8904_SetRecord (*wm8904_handle_t * handle, uint32_t recordSource*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>recordSource</i>	record source , can be a value of kCODEC_ModuleRecordSourceDifferential-Line, kCODEC_ModuleRecordSourceDifferentialMic, kCODEC_ModuleRecord-SourceSingleEndMic, kCODEC_ModuleRecordSourceDigitalMic.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

40.8.5.22 status_t WM8904_SetRecordChannel (*wm8904_handle_t * handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>leftRecord-Channel</i>	channel number of left record channel when using differential source, channel number of single end left channel when using single end source, channel number of digital mic when using digital mic source.
<i>rightRecord-Channel</i>	channel number of right record channel when using differential source, channel number of single end right channel when using single end source.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

40.8.5.23 status_t WM8904_SetPlay (**wm8904_handle_t * handle, uint32_t playSource**)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>playSource</i>	play source , can be a value of kCODEC_ModuleHeadphoneSourcePGA, kCODEC_ModuleHeadphoneSourceDAC, kCODEC_ModuleLineoutSourcePGA, kCODEC_ModuleLineoutSourceDAC.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

40.8.5.24 status_t WM8904_SetChannelMute (**wm8904_handle_t * handle, uint32_t channel, bool isMute**)

Parameters

<i>handle</i>	codec handle structure.
<i>channel</i>	codec module name.
<i>isMute</i>	true is mute, false unmute.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

40.8.6 WM8904 Adapter

40.8.6.1 Overview

The wm8904 adapter provides a codec unify control interface.

Macros

- #define `HAL_CODEC_WM8904_HANDLER_SIZE` (`WM8904_I2C_HANDLER_SIZE + 4`)
codec handler size

Functions

- `status_t HAL_CODEC_WM8904_Init` (void *handle, void *config)
Codec initialization.
- `status_t HAL_CODEC_WM8904_Deinit` (void *handle)
Codec de-initilization.
- `status_t HAL_CODEC_WM8904_SetFormat` (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
set audio data format.
- `status_t HAL_CODEC_WM8904_SetVolume` (void *handle, uint32_t playChannel, uint32_t volume)
set audio codec module volume.
- `status_t HAL_CODEC_WM8904_SetMute` (void *handle, uint32_t playChannel, bool isMute)
set audio codec module mute.
- `status_t HAL_CODEC_WM8904_SetPower` (void *handle, uint32_t module, bool powerOn)
set audio codec module power.
- `status_t HAL_CODEC_WM8904_SetRecord` (void *handle, uint32_t recordSource)
codec set record source.
- `status_t HAL_CODEC_WM8904_SetRecordChannel` (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
codec set record channel.
- `status_t HAL_CODEC_WM8904_SetPlay` (void *handle, uint32_t playSource)
codec set play source.
- `status_t HAL_CODEC_WM8904_ModuleControl` (void *handle, uint32_t cmd, uint32_t data)
codec module control.
- static `status_t HAL_CODEC_Init` (void *handle, void *config)
Codec initilization.
- static `status_t HAL_CODEC_Deinit` (void *handle)
Codec de-initilization.
- static `status_t HAL_CODEC_SetFormat` (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
set audio data format.
- static `status_t HAL_CODEC_SetVolume` (void *handle, uint32_t playChannel, uint32_t volume)
set audio codec module volume.
- static `status_t HAL_CODEC_SetMute` (void *handle, uint32_t playChannel, bool isMute)
set audio codec module mute.
- static `status_t HAL_CODEC_SetPower` (void *handle, uint32_t module, bool powerOn)

- static `status_t HAL_CODEC_SetRecord` (void *handle, uint32_t recordSource)
codec set record source.
- static `status_t HAL_CODEC_SetRecordChannel` (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
codec set record channel.
- static `status_t HAL_CODEC_SetPlay` (void *handle, uint32_t playSource)
codec set play source.
- static `status_t HAL_CODEC_ModuleControl` (void *handle, uint32_t cmd, uint32_t data)
codec module control.

40.8.6.2 Function Documentation

40.8.6.2.1 `status_t HAL_CODEC_WM8904_Init(void * handle, void * config)`

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configuration.

Returns

kStatus_Success is success, else initial failed.

40.8.6.2.2 `status_t HAL_CODEC_WM8904_Deinit(void * handle)`

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus_Success is success, else de-initial failed.

40.8.6.2.3 `status_t HAL_CODEC_WM8904_SetFormat(void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus_Success is success, else configure failed.

40.8.6.2.4 status_t HAL_CODEC_WM8904_SetVolume (void * *handle*, uint32_t *playChannel*, uint32_t *volume*)

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

40.8.6.2.5 status_t HAL_CODEC_WM8904_SetMute (void * *handle*, uint32_t *playChannel*, bool *isMute*)

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>isMute</i>	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

40.8.6.2.6 status_t HAL_CODEC_WM8904_SetPower (void * *handle*, uint32_t *module*, bool *powerOn*)

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

40.8.6.2.7 status_t HAL_CODEC_WM8904_SetRecord (void * *handle*, uint32_t *recordSource*)

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

40.8.6.2.8 status_t HAL_CODEC_WM8904_SetRecordChannel (void * *handle*, uint32_t *leftRecordChannel*, uint32_t *rightRecordChannel*)

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

40.8.6.2.9 status_t HAL_CODEC_WM8904_SetPlay (void * *handle*, uint32_t *playSource*)

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

40.8.6.2.10 status_t HAL_CODEC_WM8904_ModuleControl (void * *handle*, uint32_t *cmd*, uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference _codec_module_ctrl_cmd.
<i>data</i>	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

40.8.6.2.11 static status_t HAL_CODEC_Init (void * *handle*, void * *config*) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configuration.

Returns

kStatus_Success is success, else initial failed.

40.8.6.2.12 static status_t HAL_CODEC_Deinit (void * *handle*) [inline], [static]

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus_Success is success, else de-initial failed.

40.8.6.2.13 static status_t HAL_CODEC_SetFormat(void * *handle*, uint32_t *mclk*, uint32_t *sampleRate*, uint32_t *bitWidth*) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus_Success is success, else configure failed.

40.8.6.2.14 static status_t HAL_CODEC_SetVolume(void * *handle*, uint32_t *playChannel*, uint32_t *volume*) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

40.8.6.2.15 static status_t HAL_CODEC_SetMute(void * *handle*, uint32_t *playChannel*, bool *isMute*) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>isMute</i>	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

40.8.6.2.16 static status_t HAL_CODEC_SetPower (void * *handle*, uint32_t *module*, bool *powerOn*) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

40.8.6.2.17 static status_t HAL_CODEC_SetRecord (void * *handle*, uint32_t *recordSource*) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

40.8.6.2.18 static status_t HAL_CODEC_SetRecordChannel (void * *handle*, uint32_t *leftRecordChannel*, uint32_t *rightRecordChannel*) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

40.8.6.2.19 static status_t HAL_CODEC_SetPlay (void * *handle*, uint32_t *playSource*) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

40.8.6.2.20 static status_t HAL_CODEC_ModuleControl (void * *handle*, uint32_t *cmd*, uint32_t *data*) [inline], [static]

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference _codec_module_ctrl_cmd.
<i>data</i>	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

Chapter 41

Serial Manager

41.1 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](#)

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_MANAGER_NON_BLOCKING_MODE](#) (0U)
Enable or disable serial manager non-blocking mode (1 - enable, 0 - disable)
- #define [SERIAL_MANAGER_RING_BUFFER_FLOWCONTROL](#) (0U)
Enable or ring buffer flow control (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_UART](#) (0U)
Enable or disable uart port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_UART_DMA](#) (0U)
Enable or disable uart dma 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_VIRTUAL](#) (0U)
Enable or disable USB CDC virtual port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_RPMSG](#) (0U)
Enable or disable rpmsg port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_SPI_MASTER](#) (0U)
Enable or disable SPI Master port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_SPI_SLAVE](#) (0U)
Enable or disable SPI Slave port (1 - enable, 0 - disable)
- #define [SERIAL_MANAGER_TASK_HANDLE_TX](#) (0U)
Enable or disable SerialManager_Task() handle TX to prevent recursive calling.

- #define **SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE** (1U)
Set the default delay time in ms used by SerialManager_WriteTimeDelay().
- #define **SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE** (1U)
Set the default delay time in ms used by SerialManager_ReadTimeDelay().
- #define **SERIAL_MANAGER_TASK_HANDLE_RX_AVAILABLE_NOTIFY** (0U)
Enable or disable SerialManager_Task() handle RX data available notify.
- #define **SERIAL_MANAGER_WRITE_HANDLE_SIZE** (4U)
Set serial manager write handle size.
- #define **SERIAL_MANAGER_USE_COMMON_TASK** (0U)
SERIAL_PORT_UART_HANDLE_SIZE/SERIAL_PORT_USB_CDC_HANDLE_SIZE + serial manager dedicated size.
- #define **SERIAL_MANAGER_HANDLE_SIZE** (SERIAL_MANAGER_HANDLE_SIZE_TEMP + 12U)
Definition of serial manager handle size.
- #define **SERIAL_MANAGER_HANDLE_DEFINE**(name) uint32_t name[((**SERIAL_MANAGER_HANDLE_SIZE** + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
Defines the serial manager handle.
- #define **SERIAL_MANAGER_WRITE_HANDLE_DEFINE**(name) uint32_t name[((**SERIAL_MANAGER_WRITE_HANDLE_SIZE** + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
Defines the serial manager write handle.
- #define **SERIAL_MANAGER_READ_HANDLE_DEFINE**(name) uint32_t name[((**SERIAL_MANAGER_READ_HANDLE_SIZE** + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
Defines the serial manager read handle.
- #define **SERIAL_MANAGER_TASK_PRIORITY** (2U)
Macro to set serial manager task priority.
- #define **SERIAL_MANAGER_TASK_STACK_SIZE** (1000U)
Macro to set serial manager task stack size.

Typedefs

- typedef void * **serial_handle_t**
The handle of the serial manager module.
- typedef void * **serial_write_handle_t**
The write handle of the serial manager module.
- typedef void * **serial_read_handle_t**
The read handle of the serial manager module.
- typedef void(* **serial_manager_callback_t**)(void *callbackParam, **serial_manager_callback_message_t** *message, **serial_manager_status_t** status)
serial manager callback function
- typedef void(* **serial_manager_lowpower_critical_callback_t**)(void)
serial manager Lowpower Critical callback function

Enumerations

- enum `serial_port_type_t` {

 `kSerialPort_None` = 0U,

 `kSerialPort_Uart` = 1U,

 `kSerialPort_UsbCdc`,

 `kSerialPort_Swo`,

 `kSerialPort_Virtual`,

 `kSerialPort_Rpmsg`,

 `kSerialPort_UartDma`,

 `kSerialPort_SpiMaster`,

 `kSerialPort_SpiSlave` }

 serial port type
- enum `serial_manager_type_t` {

 `kSerialManager_NonBlocking` = 0x0U,

 `kSerialManager_Blocking` = 0x8F41U }

 serial manager 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`,

 `kStatus_SerialManager_NotConnected` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 7)` }

 serial manager error code

Functions

- `serial_manager_status_t SerialManager_Init (serial_handle_t serialHandle, const 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.
- `void SerialManager_SetLowpowerCriticalCb (const serial_manager_lowpower_critical_CBs_t *pfCallback)`
This function performs initialization of the callbacks structure used to disable lowpower when serial manager is active.

41.2 Data Structure Documentation

41.2.1 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.
- `serial_manager_type_t blockType`
Serial manager port type.
- `void * portConfig`
Serial port configuration.

Field Documentation

(1) `uint8_t* serial_manager_config_t::ringBuffer`

Besides, the memory space cannot be free during the lifetime of the serial manager module.

41.2.2 struct serial_manager_callback_message_t

Data Fields

- `uint8_t * buffer`
Transferred buffer.
- `uint32_t length`
Transferred data length.

41.3 Macro Definition Documentation

41.3.1 #define SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE (1U)

41.3.2 #define SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE (1U)

41.3.3 #define SERIAL_MANAGER_USE_COMMON_TASK (0U)

Macro to determine whether use common task.

41.3.4 #define SERIAL_MANAGER_HANDLE_SIZE (SERIAL_MANAGER_HANDLE_SIZE_TEMP + 12U)

**41.3.5 #define SERIAL_MANAGER_HANDLE_DEFINE(*name*) uint32_t
name[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) /
 sizeof(uint32_t))]**

This macro is used to define a 4 byte aligned serial manager handle. Then use "(serial_handle_t)*name*" to get the serial manager handle.

The macro should be global and could be optional. You could also define serial manager handle by yourself.

This is an example,

```
* SERIAL_MANAGER_HANDLE_DEFINE(serialManagerHandle);
*
```

Parameters

<i>name</i>	The name string of the serial manager handle.
-------------	---

**41.3.6 #define SERIAL_MANAGER_WRITE_HANDLE_DEFINE(*name*) uint32_t
name[((SERIAL_MANAGER_WRITE_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]**

This macro is used to define a 4 byte aligned serial manager write handle. Then use "(serial_write_handle_t)*name*" to get the serial manager write handle.

The macro should be global and could be optional. You could also define serial manager write handle by yourself.

This is an example,

```
* SERIAL_MANAGER_WRITE_HANDLE_DEFINE(serialManagerwriteHandle);
*
```

Parameters

<i>name</i>	The name string of the serial manager write handle.
-------------	---

**41.3.7 #define SERIAL_MANAGER_READ_HANDLE_DEFINE(*name*) uint32_t
*name[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) /
sizeof(uint32_t))]***

This macro is used to define a 4 byte aligned serial manager read handle. Then use "(serial_read_handle_t)*name*" to get the serial manager read handle.

The macro should be global and could be optional. You could also define serial manager read handle by yourself.

This is an example,

```
* SERIAL_MANAGER_READ_HANDLE_DEFINE(serialManagerReadHandle);
*
```

Parameters

<i>name</i>	The name string of the serial manager read handle.
-------------	--

41.3.8 #define SERIAL_MANAGER_TASK_PRIORITY (2U)

41.3.9 #define SERIAL_MANAGER_TASK_STACK_SIZE (1000U)

41.4 Enumeration Type Documentation

41.4.1 enum serial_port_type_t

Enumerator

- kSerialPort_None* Serial port is none.
- kSerialPort_Uart* Serial port UART.
- kSerialPort_UsbCdc* Serial port USB CDC.
- kSerialPort_Swo* Serial port SWO.
- kSerialPort_Virtual* Serial port Virtual.
- kSerialPort_Rpmsg* Serial port RPMSG.
- kSerialPort_UartDma* Serial port UART DMA.

kSerialPort_SpiMaster Serial port SPIMASTER.

kSerialPort_SpiSlave Serial port SPISLAVE.

41.4.2 enum serial_manager_type_t

Enumerator

kSerialManager_NonBlocking None blocking handle.

kSerialManager_Blocking Blocking handle.

41.4.3 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.

kStatus_SerialManager_NotConnected The host is not connected.

41.5 Function Documentation

41.5.1 serial_manager_status_t SerialManager_Init (serial_handle_t *serialHandle*, const 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 three types of serial port, UART (includes UART, USART, LPSCI, LPUART, etc), USB CDC and swo. Please refer to [serial_port_type_t](#) for serial port setting. These three 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 SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
* 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;
*   uartConfig.enableRxRTS = 0;
*   uartConfig.enableTxCTS = 0;
*   config.portConfig = &uartConfig;
*   SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*

```

For USB CDC,

```

*   #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
*   static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
*   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((serial_handle_t)s_serialHandle, &config);
*

```

Parameters

<i>serialHandle</i>	Pointer to point to a memory space of size SERIAL_MANAGER_HANDLE_SIZE allocated by the caller. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SERIAL_MANAGER_HANDLE_DEFINE(serialHandle) ; or <code>uint32_t serialHandle[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>
<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.

41.5.2 **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.

41.5.3 **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. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.
<i>writeHandle</i>	The serial manager module writing handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SERIAL_MANAGER_WRITE_HANDLE_DEFINE(writeHandle) ; or <code>uint32_t writeHandle[((SERIAL_MANAGER_WRITE_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>

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 SERIAL_MANAGER_WRITE_HANDLE_DEFINE(s_serialWriteHandle1);
* static uint8_t s_nonBlockingWelcome1[] = "This is non-blocking writing log for task1!\r\n";
* SerialManager_OpenWriteHandle((serial_handle_t)serialHandle
, (serial_write_handle_t)s_serialWriteHandle1);
* SerialManager_InstallTxCallback((serial_write_handle_t)s_serialWriteHandle1,
, Task1_SerialManagerTxCallback,
s_serialWriteHandle1);
* SerialManager_WriteNonBlocking((serial_write_handle_t)s_serialWriteHandle1,
s_nonBlockingWelcome1,
sizeof(s_nonBlockingWelcome1) - 1U);
*
```

For task 2,

```
* static SERIAL_MANAGER_WRITE_HANDLE_DEFINE(s_serialWriteHandle2);
* static uint8_t s_nonBlockingWelcome2[] = "This is non-blocking writing log for task2!\r\n";
* SerialManager_OpenWriteHandle((serial_handle_t)serialHandle
, (serial_write_handle_t)s_serialWriteHandle2);
* SerialManager_InstallTxCallback((serial_write_handle_t)s_serialWriteHandle2,
, Task2_SerialManagerTxCallback,
s_serialWriteHandle2);
* SerialManager_WriteNonBlocking((serial_write_handle_t)s_serialWriteHandle2,
s_nonBlockingWelcome2,
sizeof(s_nonBlockingWelcome2) - 1U);
*
```

41.5.4 serial_manager_status_t SerialManager_CloseWriteHandle (serial_write_handle_t *writeHandle*)

This function Closes a writing handle for the serial manager module.

Parameters

<i>writeHandle</i>	The serial manager module writing handle pointer.
--------------------	---

Return values

<i>kStatus_SerialManager_-Success</i>	The writing handle is closed.
---------------------------------------	-------------------------------

41.5.5 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. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.
<i>readHandle</i>	The serial manager module reading handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SERIAL_MANAGER_READ_HANDLE_DEFINE(readHandle) ; or <code>uint32_t readHandle[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>

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 SERIAL_MANAGER_READ_HANDLE_DEFINE(s_serialReadHandle);
* SerialManager_OpenReadHandle((serial_handle_t)serialHandle,
*     (serial_read_handle_t)s_serialReadHandle);
* static uint8_t s_nonBlockingBuffer[64];
* SerialManager_InstallRxCallback((serial_read_handle_t)s_serialReadHandle,
*     APP_SerialManagerRxCallback,
*     s_serialReadHandle);
* SerialManager_ReadNonBlocking((serial_read_handle_t)s_serialReadHandle,
*     s_nonBlockingBuffer,
*     sizeof(s_nonBlockingBuffer));
*
```

41.5.6 **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.
---------------------------------------	-------------------------------

41.5.7 `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.
<i>kStatus_SerialManager_-Error</i>	An error occurred.

41.5.8 `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

<code>readHandle</code>	The serial manager module handle pointer.
<code>buffer</code>	Start address of the data to store the received data.
<code>length</code>	The length of the data to be received.

Return values

<code>kStatus_SerialManager_-Success</code>	Successfully received all data.
<code>kStatus_SerialManager_-Busy</code>	Previous transmission still not finished; data not all received yet.
<code>kStatus_SerialManager_-Error</code>	An error occurred.

41.5.9 `serial_manager_status_t SerialManager_EnterLowpower (serial_handle_t serialHandle)`

This function is used to prepare to enter low power consumption.

Parameters

<code>serialHandle</code>	The serial manager module handle pointer.
---------------------------	---

Return values

<code>kStatus_SerialManager_-Success</code>	Successful operation.
---	-----------------------

41.5.10 `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.
---------------------------------------	-----------------------

41.5.11 void SerialManager_SetLowpowerCriticalCb (const serial_manager_lowpower_critical_CBs_t * *pfCallback*)

Parameters

<i>pfCallback</i>	Pointer to the function structure used to allow/disable lowpower.
-------------------	---

41.6 Serial Port Uart

41.6.1 Overview

Macros

- #define **SERIAL_PORT_UART_DMA_RECEIVE_DATA_LENGTH** (64U)
serial port uart handle size
- #define **SERIAL_USE_CONFIGURE_STRUCTURE** (0U)
Enable or disable the configure structure pointer.

Enumerations

- enum **serial_port_uart_parity_mode_t** {

 kSerialManager_UartParityDisabled = 0x0U,

 kSerialManager_UartParityEven = 0x2U,

 kSerialManager_UartParityOdd = 0x3U }

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

41.6.2 Enumeration Type Documentation

41.6.2.1 enum serial_port_uart_parity_mode_t

Enumerator

kSerialManager_UartParityDisabled Parity disabled.
kSerialManager_UartParityEven Parity even enabled.
kSerialManager_UartParityOdd Parity odd enabled.

41.6.2.2 enum serial_port_uart_stop_bit_count_t

Enumerator

kSerialManager_UartOneStopBit One stop bit.
kSerialManager_UartTwoStopBit Two stop bits.

41.7 Serial Port USB

41.7.1 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](#) (72U)
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.

41.7.2 Data Structure Documentation

41.7.2.1 struct serial_port_usb_cdc_config_t

Data Fields

- `serial_port_usb_cdc_controller_index_t controllerIndex`
controller index

41.7.3 Enumeration Type Documentation

41.7.3.1 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.

41.7.4 USB Device Configuration

41.8 Serial Port SWO

41.8.1 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

41.8.2 Data Structure Documentation

41.8.2.1 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.

41.8.3 Enumeration Type Documentation

41.8.3.1 enum `serial_port_swo_protocol_t`

Enumerator

`kSerialManager_SwoProtocolManchester` SWO Manchester protocol.
`kSerialManager_SwoProtocolNrz` SWO UART/NRZ protocol.

41.8.4 CODEC Adapter

41.8.4.1 Overview

Enumerations

- enum {
 kCODEC_WM8904,
 kCODEC_WM8960,
 kCODEC_WM8524,
 kCODEC_SGTL5000,
 kCODEC_DA7212,
 kCODEC_CS42888,
 kCODEC_CS42448,
 kCODEC_AK4497,
 kCODEC_AK4458,
 kCODEC_TFA9XXX,
 kCODEC_TFA9896 }
 codec type

41.8.4.2 Enumeration Type Documentation

41.8.4.2.1 anonymous enum

Enumerator

kCODEC_WM8904 wm8904
kCODEC_WM8960 wm8960
kCODEC_WM8524 wm8524
kCODEC_SGTL5000 sgtl5000
kCODEC_DA7212 da7212
kCODEC_CS42888 CS42888.
kCODEC_CS42448 CS42448.
kCODEC_AK4497 AK4497.
kCODEC_AK4458 ak4458
kCODEC_TFA9XXX tfa9xxx
kCODEC_TFA9896 tfa9896

How to Reach Us:

Home Page:

nxp.com

Web Support:

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.

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.

© 2021 NXP B.V.

