
MCUXpresso SDK API Reference Manual

NXP Semiconductors

Document Number: MCUXSDKAPIRM
Rev. 0
Aug 2020



Contents

Chapter [Introduction](#)

Chapter [Trademarks](#)

Chapter [Architectural Overview](#)

Chapter [Driver errors status](#)

Chapter [Deprecated List](#)

Chapter [Clock Driver](#)

6.1	Overview	15
6.2	Data Structure Documentation	23
6.2.1	struct sim_clock_config_t	23
6.2.2	struct oscr_config_t	23
6.2.3	struct osc_config_t	24
6.2.4	struct mcg_pll_config_t	24
6.2.5	struct mcg_config_t	25
6.3	Macro Definition Documentation	26
6.3.1	MCG_CONFIG_CHECK_PARAM	26
6.3.2	FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL	26
6.3.3	FSL_CLOCK_DRIVER_VERSION	27
6.3.4	MCG_INTERNAL_IRC_48M	27
6.3.5	DMAMUX_CLOCKS	27
6.3.6	RTC_CLOCKS	27
6.3.7	ENET_CLOCKS	27
6.3.8	PORT_CLOCKS	27
6.3.9	SAI_CLOCKS	27
6.3.10	FLEXBUS_CLOCKS	28
6.3.11	EWM_CLOCKS	28
6.3.12	PIT_CLOCKS	28
6.3.13	DSPI_CLOCKS	28
6.3.14	LPTMR_CLOCKS	28
6.3.15	SDHC_CLOCKS	29

Contents

Section Number	Title	Page Number
6.3.16	FTM_CLOCKS	29
6.3.17	EDMA_CLOCKS	29
6.3.18	FLEXCAN_CLOCKS	29
6.3.19	DAC_CLOCKS	29
6.3.20	ADC16_CLOCKS	30
6.3.21	SYSMPU_CLOCKS	30
6.3.22	VREF_CLOCKS	30
6.3.23	CMT_CLOCKS	30
6.3.24	UART_CLOCKS	30
6.3.25	RNGA_CLOCKS	31
6.3.26	CRC_CLOCKS	31
6.3.27	I2C_CLOCKS	31
6.3.28	PDB_CLOCKS	31
6.3.29	FTF_CLOCKS	31
6.3.30	CMP_CLOCKS	32
6.3.31	SYS_CLK	32
6.4	Enumeration Type Documentation	32
6.4.1	clock_name_t	32
6.4.2	clock_usb_src_t	32
6.4.3	clock_ip_name_t	33
6.4.4	osc_mode_t	33
6.4.5	_osc_cap_load	33
6.4.6	_oscer_enable_mode	33
6.4.7	mcg_fll_src_t	33
6.4.8	mcg_irc_mode_t	33
6.4.9	mcg_dmx32_t	34
6.4.10	mcg_drs_t	34
6.4.11	mcg_pll_ref_src_t	34
6.4.12	mcg_clkout_src_t	34
6.4.13	mcg_atm_select_t	34
6.4.14	mcg_oscsel_t	35
6.4.15	mcg_pll_clk_select_t	35
6.4.16	mcg_monitor_mode_t	35
6.4.17	anonymous enum	35
6.4.18	anonymous enum	35
6.4.19	anonymous enum	36
6.4.20	anonymous enum	36
6.4.21	mcg_mode_t	36
6.5	Function Documentation	36
6.5.1	CLOCK_EnableClock	36
6.5.2	CLOCK_DisableClock	37
6.5.3	CLOCK_SetEr32kClock	37
6.5.4	CLOCK_SetSdhc0Clock	37

Contents

Section Number	Title	Page Number
6.5.5	CLOCK_SetEnetTime0Clock	37
6.5.6	CLOCK_SetRmii0Clock	37
6.5.7	CLOCK_SetTraceClock	38
6.5.8	CLOCK_SetPllFllSelClock	38
6.5.9	CLOCK_SetClkOutClock	38
6.5.10	CLOCK_SetRtcClkOutClock	38
6.5.11	CLOCK_EnableUsbfs0Clock	38
6.5.12	CLOCK_DisableUsbfs0Clock	39
6.5.13	CLOCK_SetOutDiv	39
6.5.14	CLOCK_GetFreq	39
6.5.15	CLOCK_GetCoreSysClkFreq	40
6.5.16	CLOCK_GetPlatClkFreq	40
6.5.17	CLOCK_GetBusClkFreq	40
6.5.18	CLOCK_GetFlexBusClkFreq	40
6.5.19	CLOCK_GetFlashClkFreq	40
6.5.20	CLOCK_GetPllFllSelClkFreq	41
6.5.21	CLOCK_GetEr32kClkFreq	41
6.5.22	CLOCK_GetOsc0ErClkFreq	41
6.5.23	CLOCK_SetSimConfig	41
6.5.24	CLOCK_SetSimSafeDivs	41
6.5.25	CLOCK_GetOutClkFreq	41
6.5.26	CLOCK_GetFllFreq	42
6.5.27	CLOCK_GetInternalRefClkFreq	42
6.5.28	CLOCK_GetFixedFreqClkFreq	42
6.5.29	CLOCK_GetPll0Freq	42
6.5.30	CLOCK_SetLowPowerEnable	42
6.5.31	CLOCK_SetInternalRefClkConfig	43
6.5.32	CLOCK_SetExternalRefClkConfig	43
6.5.33	CLOCK_SetFllExtRefDiv	44
6.5.34	CLOCK_EnablePll0	44
6.5.35	CLOCK_DisablePll0	44
6.5.36	CLOCK_CalcPllDiv	44
6.5.37	CLOCK_SetOsc0MonitorMode	45
6.5.38	CLOCK_SetRtcOscMonitorMode	45
6.5.39	CLOCK_SetPll0MonitorMode	45
6.5.40	CLOCK_GetStatusFlags	45
6.5.41	CLOCK_ClearStatusFlags	46
6.5.42	OSC_SetExtRefClkConfig	46
6.5.43	OSC_SetCapLoad	47
6.5.44	CLOCK_InitOsc0	47
6.5.45	CLOCK_DeinitOsc0	47
6.5.46	CLOCK_SetXtal0Freq	47
6.5.47	CLOCK_SetXtal32Freq	48
6.5.48	CLOCK_SetSlowIrcFreq	48
6.5.49	CLOCK_SetFastIrcFreq	48

Contents

Section Number	Title	Page Number
6.5.50	CLOCK_TrimInternalRefClk	48
6.5.51	CLOCK_GetMode	49
6.5.52	CLOCK_SetFeiMode	49
6.5.53	CLOCK_SetFeeMode	50
6.5.54	CLOCK_SetFbiMode	50
6.5.55	CLOCK_SetFbeMode	51
6.5.56	CLOCK_SetBlpiMode	52
6.5.57	CLOCK_SetBlpeMode	52
6.5.58	CLOCK_SetPbeMode	52
6.5.59	CLOCK_SetPeeMode	53
6.5.60	CLOCK_ExternalModeToFbeModeQuick	53
6.5.61	CLOCK_InternalModeToFbiModeQuick	54
6.5.62	CLOCK_BootToFeiMode	54
6.5.63	CLOCK_BootToFeeMode	55
6.5.64	CLOCK_BootToBlpiMode	55
6.5.65	CLOCK_BootToBlpeMode	56
6.5.66	CLOCK_BootToPeeMode	56
6.5.67	CLOCK_SetMcgConfig	57
6.6	Variable Documentation	57
6.6.1	g_xtal0Freq	57
6.6.2	g_xtal32Freq	58
6.7	Multipurpose Clock Generator (MCG)	59
6.7.1	Function description	59
6.7.2	Typical use case	61
6.7.3	Code Configuration Option	64
Chapter	ADC16: 16-bit SAR Analog-to-Digital Converter Driver	
7.1	Overview	65
7.2	Typical use case	65
7.2.1	Polling Configuration	65
7.2.2	Interrupt Configuration	65
7.3	Data Structure Documentation	68
7.3.1	struct adc16_config_t	68
7.3.2	struct adc16_hardware_compare_config_t	69
7.3.3	struct adc16_channel_config_t	69
7.4	Macro Definition Documentation	70
7.4.1	FSL_ADC16_DRIVER_VERSION	70
7.5	Enumeration Type Documentation	70
7.5.1	_adc16_channel_status_flags	70

Contents

Section Number	Title	Page Number
7.5.2	_adc16_status_flags	70
7.5.3	adc16_channel_mux_mode_t	70
7.5.4	adc16_clock_divider_t	71
7.5.5	adc16_resolution_t	71
7.5.6	adc16_clock_source_t	71
7.5.7	adc16_long_sample_mode_t	71
7.5.8	adc16_reference_voltage_source_t	72
7.5.9	adc16_hardware_average_mode_t	72
7.5.10	adc16_hardware_compare_mode_t	72
7.6	Function Documentation	72
7.6.1	ADC16_Init	72
7.6.2	ADC16_Deinit	73
7.6.3	ADC16_GetDefaultConfig	73
7.6.4	ADC16_DoAutoCalibration	73
7.6.5	ADC16_SetOffsetValue	74
7.6.6	ADC16_EnableDMA	74
7.6.7	ADC16_EnableHardwareTrigger	74
7.6.8	ADC16_SetChannelMuxMode	75
7.6.9	ADC16_SetHardwareCompareConfig	75
7.6.10	ADC16_SetHardwareAverage	75
7.6.11	ADC16_GetStatusFlags	76
7.6.12	ADC16_ClearStatusFlags	76
7.6.13	ADC16_SetChannelConfig	76
7.6.14	ADC16_GetChannelConversionValue	78
7.6.15	ADC16_GetChannelStatusFlags	78
Chapter	CMP: Analog Comparator Driver	
8.1	Overview	79
8.2	Typical use case	79
8.2.1	Polling Configuration	79
8.2.2	Interrupt Configuration	79
8.3	Data Structure Documentation	81
8.3.1	struct cmp_config_t	81
8.3.2	struct cmp_filter_config_t	81
8.3.3	struct cmp_dac_config_t	82
8.4	Macro Definition Documentation	82
8.4.1	FSL_CMP_DRIVER_VERSION	82
8.5	Enumeration Type Documentation	82
8.5.1	_cmp_interrupt_enable	82

Contents

Section Number	Title	Page Number
8.5.2	_cmp_status_flags	82
8.5.3	cmp_hysteresis_mode_t	83
8.5.4	cmp_reference_voltage_source_t	83
8.6	Function Documentation	83
8.6.1	CMP_Init	83
8.6.2	CMP_Deinit	83
8.6.3	CMP_Enable	84
8.6.4	CMP_GetDefaultConfig	84
8.6.5	CMP_SetInputChannels	84
8.6.6	CMP_EnableDMA	85
8.6.7	CMP_EnableWindowMode	85
8.6.8	CMP_EnablePassThroughMode	85
8.6.9	CMP_SetFilterConfig	86
8.6.10	CMP_SetDACConfig	86
8.6.11	CMP_EnableInterrupts	86
8.6.12	CMP_DisableInterrupts	86
8.6.13	CMP_GetStatusFlags	87
8.6.14	CMP_ClearStatusFlags	87
Chapter	CMT: Carrier Modulator Transmitter Driver	
9.1	Overview	89
9.2	Clock formulas	89
9.3	Typical use case	89
9.4	Data Structure Documentation	91
9.4.1	struct cmt_modulate_config_t	91
9.4.2	struct cmt_config_t	92
9.5	Macro Definition Documentation	93
9.5.1	FSL_CMT_DRIVER_VERSION	93
9.6	Enumeration Type Documentation	93
9.6.1	cmt_mode_t	93
9.6.2	cmt_primary_clkdiv_t	93
9.6.3	cmt_second_clkdiv_t	93
9.6.4	cmt_infrared_output_polarity_t	94
9.6.5	cmt_infrared_output_state_t	94
9.6.6	_cmt_interrupt_enable	94
9.7	Function Documentation	94
9.7.1	CMT_GetDefaultConfig	94
9.7.2	CMT_Init	95

Contents

Section Number	Title	Page Number
9.7.3	CMT_Deinit	95
9.7.4	CMT_SetMode	95
9.7.5	CMT_GetMode	95
9.7.6	CMT_GetCMTFrequency	96
9.7.7	CMT_SetCarrirGenerateCountOne	96
9.7.8	CMT_SetCarrirGenerateCountTwo	97
9.7.9	CMT_SetModulateMarkSpace	97
9.7.10	CMT_EnableExtendedSpace	97
9.7.11	CMT_SetIroState	98
9.7.12	CMT_EnableInterrupts	98
9.7.13	CMT_DisableInterrupts	98
9.7.14	CMT_GetStatusFlags	99

Chapter Common Driver

10.1	Overview	101
10.2	Macro Definition Documentation	105
10.2.1	MAKE_STATUS	105
10.2.2	MAKE_VERSION	105
10.2.3	FSL_COMMON_DRIVER_VERSION	105
10.2.4	DEBUG_CONSOLE_DEVICE_TYPE_NONE	105
10.2.5	DEBUG_CONSOLE_DEVICE_TYPE_UART	105
10.2.6	DEBUG_CONSOLE_DEVICE_TYPE_LPUART	105
10.2.7	DEBUG_CONSOLE_DEVICE_TYPE_LPSCI	105
10.2.8	DEBUG_CONSOLE_DEVICE_TYPE_USBCDC	105
10.2.9	DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM	105
10.2.10	DEBUG_CONSOLE_DEVICE_TYPE_IUART	105
10.2.11	DEBUG_CONSOLE_DEVICE_TYPE_VUSART	105
10.2.12	DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART	105
10.2.13	DEBUG_CONSOLE_DEVICE_TYPE_SWO	105
10.2.14	ARRAY_SIZE	105
10.3	Typedef Documentation	105
10.3.1	status_t	105
10.4	Enumeration Type Documentation	105
10.4.1	_status_groups	105
10.4.2	anonymous enum	108
10.5	Function Documentation	108
10.5.1	EnableIRQ	108
10.5.2	DisableIRQ	109
10.5.3	DisableGlobalIRQ	109
10.5.4	EnableGlobalIRQ	109

Contents

Section Number	Title	Page Number
10.5.5	SDK_Malloc	110
10.5.6	SDK_Free	110
10.5.7	SDK_DelayAtLeastUs	110
Chapter	CRC: Cyclic Redundancy Check Driver	
11.1	Overview	111
11.2	CRC Driver Initialization and Configuration	111
11.3	CRC Write Data	111
11.4	CRC Get Checksum	111
11.5	Comments about API usage in RTOS	112
11.6	Data Structure Documentation	113
11.6.1	struct crc_config_t	113
11.7	Macro Definition Documentation	114
11.7.1	FSL_CRC_DRIVER_VERSION	114
11.7.2	CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT	114
11.8	Enumeration Type Documentation	114
11.8.1	crc_bits_t	114
11.8.2	crc_result_t	114
11.9	Function Documentation	114
11.9.1	CRC_Init	114
11.9.2	CRC_Deinit	115
11.9.3	CRC_GetDefaultConfig	115
11.9.4	CRC_WriteData	115
11.9.5	CRC_Get32bitResult	116
11.9.6	CRC_Get16bitResult	116
Chapter	DAC: Digital-to-Analog Converter Driver	
12.1	Overview	117
12.2	Typical use case	117
12.2.1	Working as a basic DAC without the hardware buffer feature	117
12.2.2	Working with the hardware buffer	117
12.3	Data Structure Documentation	119
12.3.1	struct dac_config_t	119
12.3.2	struct dac_buffer_config_t	119

Contents

Section Number	Title	Page Number
12.4	Macro Definition Documentation	120
12.4.1	FSL_DAC_DRIVER_VERSION	120
12.5	Enumeration Type Documentation	120
12.5.1	_dac_buffer_status_flags	120
12.5.2	_dac_buffer_interrupt_enable	120
12.5.3	dac_reference_voltage_source_t	120
12.5.4	dac_buffer_trigger_mode_t	121
12.5.5	dac_buffer_watermark_t	121
12.5.6	dac_buffer_work_mode_t	121
12.6	Function Documentation	121
12.6.1	DAC_Init	121
12.6.2	DAC_Deinit	121
12.6.3	DAC_GetDefaultConfig	122
12.6.4	DAC_Enable	122
12.6.5	DAC_EnableBuffer	122
12.6.6	DAC_SetBufferConfig	122
12.6.7	DAC_GetDefaultBufferConfig	123
12.6.8	DAC_EnableBufferDMA	123
12.6.9	DAC_SetBufferValue	123
12.6.10	DAC_DoSoftwareTriggerBuffer	124
12.6.11	DAC_GetBufferReadPointer	125
12.6.12	DAC_SetBufferReadPointer	125
12.6.13	DAC_EnableBufferInterrupts	125
12.6.14	DAC_DisableBufferInterrupts	125
12.6.15	DAC_GetBufferStatusFlags	126
12.6.16	DAC_ClearBufferStatusFlags	126
Chapter	DMAMUX: Direct Memory Access Multiplexer Driver	
13.1	Overview	127
13.2	Typical use case	127
13.2.1	DMAMUX Operation	127
13.3	Macro Definition Documentation	127
13.3.1	FSL_DMAMUX_DRIVER_VERSION	127
13.4	Function Documentation	128
13.4.1	DMAMUX_Init	128
13.4.2	DMAMUX_Deinit	129
13.4.3	DMAMUX_EnableChannel	129
13.4.4	DMAMUX_DisableChannel	129
13.4.5	DMAMUX_SetSource	130

Contents

Section Number	Title	Page Number
13.4.6	DMAMUX_EnablePeriodTrigger	130
13.4.7	DMAMUX_DisablePeriodTrigger	130
Chapter	DSPI: Serial Peripheral Interface Driver	
14.1	Overview	131
14.2	DSPI Driver	132
14.2.1	Overview	132
14.2.2	Typical use case	132
14.2.3	Data Structure Documentation	139
14.2.4	Macro Definition Documentation	146
14.2.5	Typedef Documentation	147
14.2.6	Enumeration Type Documentation	148
14.2.7	Function Documentation	152
14.2.8	Variable Documentation	171
14.3	DSPI eDMA Driver	172
14.3.1	Overview	172
14.3.2	Data Structure Documentation	175
14.3.3	Macro Definition Documentation	176
14.3.4	Typedef Documentation	177
14.3.5	Function Documentation	177
14.3.6	Variable Documentation	184
14.4	DSPI FreeRTOS Driver	185
14.4.1	Overview	185
14.4.2	Macro Definition Documentation	185
14.4.3	Function Documentation	185
14.5	DSPI CMSIS Driver	188
14.5.1	Function groups	188
14.5.2	Typical use case	189
Chapter	eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver	
15.1	Overview	191
15.2	Typical use case	191
15.2.1	eDMA Operation	191
15.3	Data Structure Documentation	196
15.3.1	struct edma_config_t	196
15.3.2	struct edma_transfer_config_t	197
15.3.3	struct edma_channel_Preemption_config_t	197
15.3.4	struct edma_minor_offset_config_t	198

Contents

Section Number	Title	Page Number
15.3.5	struct edma_tcd_t	198
15.3.6	struct edma_handle_t	199
15.4	Macro Definition Documentation	200
15.4.1	FSL_EDMA_DRIVER_VERSION	200
15.5	Typedef Documentation	200
15.5.1	edma_callback	200
15.6	Enumeration Type Documentation	201
15.6.1	edma_transfer_size_t	201
15.6.2	edma_modulo_t	201
15.6.3	edma_bandwidth_t	202
15.6.4	edma_channel_link_type_t	202
15.6.5	anonymous enum	202
15.6.6	anonymous enum	203
15.6.7	edma_interrupt_enable_t	203
15.6.8	edma_transfer_type_t	203
15.6.9	anonymous enum	203
15.7	Function Documentation	204
15.7.1	EDMA_Init	204
15.7.2	EDMA_Deinit	204
15.7.3	EDMA_InstallTCD	204
15.7.4	EDMA_GetDefaultConfig	204
15.7.5	EDMA_ResetChannel	206
15.7.6	EDMA_SetTransferConfig	206
15.7.7	EDMA_SetMinorOffsetConfig	207
15.7.8	EDMA_SetChannelPreemptionConfig	207
15.7.9	EDMA_SetChannelLink	207
15.7.10	EDMA_SetBandWidth	208
15.7.11	EDMA_SetModulo	208
15.7.12	EDMA_EnableAutoStopRequest	209
15.7.13	EDMA_EnableChannelInterrupts	209
15.7.14	EDMA_DisableChannelInterrupts	209
15.7.15	EDMA_TcdReset	210
15.7.16	EDMA_TcdSetTransferConfig	210
15.7.17	EDMA_TcdSetMinorOffsetConfig	211
15.7.18	EDMA_TcdSetChannelLink	212
15.7.19	EDMA_TcdSetBandWidth	212
15.7.20	EDMA_TcdSetModulo	213
15.7.21	EDMA_TcdEnableAutoStopRequest	213
15.7.22	EDMA_TcdEnableInterrupts	213
15.7.23	EDMA_TcdDisableInterrupts	213
15.7.24	EDMA_EnableChannelRequest	214

Contents

Section Number	Title	Page Number
15.7.25	EDMA_DisableChannelRequest	214
15.7.26	EDMA_TriggerChannelStart	214
15.7.27	EDMA_GetRemainingMajorLoopCount	215
15.7.28	EDMA_GetErrorStatusFlags	216
15.7.29	EDMA_GetChannelStatusFlags	216
15.7.30	EDMA_ClearChannelStatusFlags	217
15.7.31	EDMA_CreateHandle	217
15.7.32	EDMA_InstallTCDDMemory	217
15.7.33	EDMA_SetCallback	219
15.7.34	EDMA_PrepareTransferConfig	219
15.7.35	EDMA_PrepareTransfer	220
15.7.36	EDMA_SubmitTransfer	220
15.7.37	EDMA_StartTransfer	221
15.7.38	EDMA_StopTransfer	221
15.7.39	EDMA_AbortTransfer	221
15.7.40	EDMA_GetUnusedTCDNumber	222
15.7.41	EDMA_GetNextTCDAddress	222
15.7.42	EDMA_HandleIRQ	222

Chapter ENET: Ethernet MAC Driver

16.1	Overview	225
16.2	Operations of Ethernet MAC Driver	225
16.2.1	MII interface Operation	225
16.2.2	MAC address filter	225
16.2.3	Other Baisc control Operations	225
16.2.4	Transactional Operation	225
16.2.5	PTP IEEE 1588 Feature Operation	226
16.3	Typical use case	226
16.3.1	ENET Initialization, receive, and transmit operations	226
16.4	Data Structure Documentation	233
16.4.1	struct enet_rx_bd_struct_t	233
16.4.2	struct enet_tx_bd_struct_t	233
16.4.3	struct enet_data_error_stats_t	234
16.4.4	struct enet_frame_info_t	234
16.4.5	struct enet_tx_dirty_ring_t	234
16.4.6	struct enet_buffer_config_t	235
16.4.7	struct enet_config_t	236
16.4.8	struct enet_tx_bd_ring_t	238
16.4.9	struct enet_rx_bd_ring_t	239
16.4.10	struct _enet_handle	239

Contents

Section Number	Title	Page Number
16.5	Macro Definition Documentation	240
16.5.1	FSL_ENET_DRIVER_VERSION	240
16.5.2	FSL_FEATURE_ENET_QUEUE	242
16.5.3	ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK	242
16.5.4	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK	242
16.5.5	ENET_BUFFDESCRIPTOR_RX_WRAP_MASK	242
16.5.6	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask	242
16.5.7	ENET_BUFFDESCRIPTOR_RX_LAST_MASK	242
16.5.8	ENET_BUFFDESCRIPTOR_RX_MISS_MASK	242
16.5.9	ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK	242
16.5.10	ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK	242
16.5.11	ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK	242
16.5.12	ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK	242
16.5.13	ENET_BUFFDESCRIPTOR_RX_CRC_MASK	242
16.5.14	ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK	242
16.5.15	ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK	242
16.5.16	ENET_BUFFDESCRIPTOR_TX_READY_MASK	242
16.5.17	ENET_BUFFDESCRIPTOR_TX_SOFTOWENER1_MASK	242
16.5.18	ENET_BUFFDESCRIPTOR_TX_WRAP_MASK	242
16.5.19	ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_MASK	242
16.5.20	ENET_BUFFDESCRIPTOR_TX_LAST_MASK	242
16.5.21	ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK	242
16.5.22	ENET_BUFFDESCRIPTOR_RX_ERR_MASK	242
16.5.23	ENET_FRAME_MAX_FRAMELEN	243
16.5.24	ENET_FIFO_MIN_RX_FULL	243
16.5.25	ENET_RX_MIN_BUFFERSIZE	243
16.5.26	ENET_PHY_MAXADDRESS	243
16.5.27	ENET_TX_INTERRUPT	243
16.5.28	ENET_RX_INTERRUPT	243
16.5.29	ENET_TS_INTERRUPT	243
16.5.30	ENET_ERR_INTERRUPT	243
16.5.31	ENET_TX_LAST_BD_FLAG	243
16.5.32	ENET_TX_TIMESTAMP_FLAG	243
16.6	Typedef Documentation	244
16.6.1	enet_callback_t	244
16.6.2	enet_isr_t	244
16.7	Enumeration Type Documentation	244
16.7.1	anonymous enum	244
16.7.2	enet_mii_mode_t	244
16.7.3	enet_mii_speed_t	244
16.7.4	enet_mii_duplex_t	244
16.7.5	enet_mii_write_t	245
16.7.6	enet_mii_read_t	245

Contents

Section Number	Title	Page Number
16.7.7	enet_special_control_flag_t	245
16.7.8	enet_interrupt_enable_t	245
16.7.9	enet_event_t	246
16.7.10	enet_tx_accelerator_t	246
16.7.11	enet_rx_accelerator_t	246
16.8	Function Documentation	247
16.8.1	ENET_GetInstance	247
16.8.2	ENET_GetDefaultConfig	247
16.8.3	ENET_Up	247
16.8.4	ENET_Init	248
16.8.5	ENET_Down	249
16.8.6	ENET_Deinit	249
16.8.7	ENET_Reset	249
16.8.8	ENET_SetMII	250
16.8.9	ENET_SetSMI	250
16.8.10	ENET_GetSMI	250
16.8.11	ENET_ReadSMIData	251
16.8.12	ENET_StartSMIRead	251
16.8.13	ENET_StartSMIWrite	251
16.8.14	ENET_SetMacAddr	252
16.8.15	ENET_GetMacAddr	252
16.8.16	ENET_AddMulticastGroup	252
16.8.17	ENET_LeaveMulticastGroup	252
16.8.18	ENET_ActiveRead	253
16.8.19	ENET_EnableSleepMode	253
16.8.20	ENET_GetAccelFunction	253
16.8.21	ENET_EnableInterrupts	254
16.8.22	ENET_DisableInterrupts	254
16.8.23	ENET_GetInterruptStatus	255
16.8.24	ENET_ClearInterruptStatus	255
16.8.25	ENET_SetRxISRHandler	255
16.8.26	ENET_SetTxISRHandler	256
16.8.27	ENET_SetErrISRHandler	256
16.8.28	ENET_SetCallback	256
16.8.29	ENET_GetRxErrBeforeReadFrame	256
16.8.30	ENET_GetRxFrameSize	257
16.8.31	ENET_ReadFrame	258
16.8.32	ENET_SendFrame	259
16.8.33	ENET_SetTxReclaim	259
16.8.34	ENET_GetRxBuffer	260
16.8.35	ENET_ReleaseRxBuffer	261
16.8.36	ENET_SendFrameZeroCopy	262
16.8.37	ENET_SetTxBuffer	262
16.8.38	ENET_TransmitIRQHandler	263

Contents

Section Number	Title	Page Number
16.8.39	ENET_ReceiveIRQHandler	263
16.8.40	ENET_ErrorIRQHandler	264
16.8.41	ENET_CommonFrame0IRQHandler	264
16.9	ENET CMSIS Driver	265
16.9.1	Typical use case	265
Chapter	EWM: External Watchdog Monitor Driver	
17.1	Overview	267
17.2	Typical use case	267
17.3	Data Structure Documentation	268
17.3.1	struct ewm_config_t	268
17.4	Macro Definition Documentation	268
17.4.1	FSL_EWM_DRIVER_VERSION	268
17.5	Enumeration Type Documentation	268
17.5.1	_ewm_interrupt_enable_t	268
17.5.2	_ewm_status_flags_t	268
17.6	Function Documentation	269
17.6.1	EWM_Init	269
17.6.2	EWM_Deinit	269
17.6.3	EWM_GetDefaultConfig	269
17.6.4	EWM_EnableInterrupts	270
17.6.5	EWM_DisableInterrupts	270
17.6.6	EWM_GetStatusFlags	270
17.6.7	EWM_Refresh	271
Chapter	C90TFS Flash Driver	
18.1	Overview	273
18.2	Ftftx FLASH Driver	274
18.2.1	Overview	274
18.2.2	Data Structure Documentation	276
18.2.3	Macro Definition Documentation	277
18.2.4	Enumeration Type Documentation	277
18.2.5	Function Documentation	278
18.3	Ftftx CACHE Driver	293
18.3.1	Overview	293
18.3.2	Data Structure Documentation	293

Contents

Section Number	Title	Page Number
18.3.3	Macro Definition Documentation	294
18.3.4	Enumeration Type Documentation	294
18.3.5	Function Documentation	294
18.4	Ftftx FLEXNVM Driver	297
18.4.1	Overview	297
18.4.2	Data Structure Documentation	299
18.4.3	Macro Definition Documentation	299
18.4.4	Enumeration Type Documentation	300
18.4.5	Function Documentation	300
18.5	ftfx feature	317
18.5.1	Overview	317
18.5.2	Macro Definition Documentation	317
18.5.3	ftfx adapter	318
18.6	ftfx controller	319
18.6.1	Overview	319
18.6.2	Data Structure Documentation	322
18.6.3	Macro Definition Documentation	324
18.6.4	Enumeration Type Documentation	324
18.6.5	Function Documentation	326
18.6.6	ftfx utilities	341
Chapter	FlexBus: External Bus Interface Driver	
19.1	Overview	343
19.2	FlexBus functional operation	343
19.3	Typical use case and example	343
19.4	Data Structure Documentation	345
19.4.1	struct flexbus_config_t	345
19.5	Macro Definition Documentation	346
19.5.1	FSL_FLEXBUS_DRIVER_VERSION	346
19.6	Enumeration Type Documentation	346
19.6.1	flexbus_port_size_t	346
19.6.2	flexbus_write_address_hold_t	346
19.6.3	flexbus_read_address_hold_t	346
19.6.4	flexbus_address_setup_t	347
19.6.5	flexbus_bytelane_shift_t	347
19.6.6	flexbus_multiplex_group1_t	347
19.6.7	flexbus_multiplex_group2_t	347

Contents

Section Number	Title	Page Number
19.6.8	flexbus_multiplex_group3_t	348
19.6.9	flexbus_multiplex_group4_t	348
19.6.10	flexbus_multiplex_group5_t	348
19.7	Function Documentation	348
19.7.1	FLEXBUS_Init	348
19.7.2	FLEXBUS_Deinit	349
19.7.3	FLEXBUS_GetDefaultConfig	349
Chapter	FlexCAN: Flex Controller Area Network Driver	
20.1	Overview	351
20.2	FlexCAN Driver	352
20.2.1	Overview	352
20.2.2	Typical use case	352
20.2.3	Data Structure Documentation	360
20.2.4	Macro Definition Documentation	365
20.2.5	Typedef Documentation	370
20.2.6	Enumeration Type Documentation	370
20.2.7	Function Documentation	373
Chapter	FTM: FlexTimer Driver	
21.1	Overview	389
21.2	Function groups	389
21.2.1	Initialization and deinitialization	389
21.2.2	PWM Operations	389
21.2.3	Input capture operations	389
21.2.4	Output compare operations	390
21.2.5	Quad decode	390
21.2.6	Fault operation	390
21.3	Register Update	390
21.4	Typical use case	391
21.4.1	PWM output	391
21.5	Data Structure Documentation	397
21.5.1	struct ftm_chnl_pwm_signal_param_t	397
21.5.2	struct ftm_chnl_pwm_config_param_t	398
21.5.3	struct ftm_dual_edge_capture_param_t	398
21.5.4	struct ftm_phase_params_t	398
21.5.5	struct ftm_fault_param_t	399
21.5.6	struct ftm_config_t	399

Contents

Section Number	Title	Page Number
21.6	Macro Definition Documentation	400
21.6.1	FSL_FTM_DRIVER_VERSION	400
21.7	Enumeration Type Documentation	400
21.7.1	ftm_chnl_t	400
21.7.2	ftm_fault_input_t	401
21.7.3	ftm_pwm_mode_t	401
21.7.4	ftm_pwm_level_select_t	401
21.7.5	ftm_output_compare_mode_t	401
21.7.6	ftm_input_capture_edge_t	401
21.7.7	ftm_dual_edge_capture_mode_t	402
21.7.8	ftm_quad_decode_mode_t	402
21.7.9	ftm_phase_polarity_t	402
21.7.10	ftm_deadtime_prescale_t	402
21.7.11	ftm_clock_source_t	402
21.7.12	ftm_clock_prescale_t	403
21.7.13	ftm_bdm_mode_t	403
21.7.14	ftm_fault_mode_t	403
21.7.15	ftm_external_trigger_t	403
21.7.16	ftm_pwm_sync_method_t	404
21.7.17	ftm_reload_point_t	404
21.7.18	ftm_interrupt_enable_t	405
21.7.19	ftm_status_flags_t	405
21.7.20	anonymous enum	406
21.8	Function Documentation	406
21.8.1	FTM_Init	406
21.8.2	FTM_Deinit	406
21.8.3	FTM_GetDefaultConfig	406
21.8.4	FTM_SetupPwm	407
21.8.5	FTM_UpdatePwmDutycycle	407
21.8.6	FTM_UpdateChnlEdgeLevelSelect	408
21.8.7	FTM_SetupPwmMode	408
21.8.8	FTM_SetupInputCapture	409
21.8.9	FTM_SetupOutputCompare	409
21.8.10	FTM_SetupDualEdgeCapture	409
21.8.11	FTM_SetupFault	410
21.8.12	FTM_EnableInterrupts	410
21.8.13	FTM_DisableInterrupts	410
21.8.14	FTM_GetEnabledInterrupts	411
21.8.15	FTM_GetStatusFlags	412
21.8.16	FTM_ClearStatusFlags	412
21.8.17	FTM_SetTimerPeriod	412
21.8.18	FTM_GetCurrentTimerCount	413
21.8.19	FTM_StartTimer	413

Contents

Section Number	Title	Page Number
21.8.20	FTM_StopTimer	413
21.8.21	FTM_SetSoftwareCtrlEnable	413
21.8.22	FTM_SetSoftwareCtrlVal	414
21.8.23	FTM_SetGlobalTimeBaseOutputEnable	414
21.8.24	FTM_SetOutputMask	414
21.8.25	FTM_SetFaultControlEnable	415
21.8.26	FTM_SetDeadTimeEnable	416
21.8.27	FTM_SetComplementaryEnable	416
21.8.28	FTM_SetInvertEnable	416
21.8.29	FTM_SetupQuadDecode	417
21.8.30	FTM_GetQuadDecoderFlags	417
21.8.31	FTM_SetQuadDecoderModuloValue	417
21.8.32	FTM_GetQuadDecoderCounterValue	418
21.8.33	FTM_ClearQuadDecoderCounterValue	418
21.8.34	FTM_SetSoftwareTrigger	418
21.8.35	FTM_SetWriteProtection	418
21.8.36	FTM_EnableDmaTransfer	419

Chapter GPIO: General-Purpose Input/Output Driver

22.1	Overview	421
22.2	Data Structure Documentation	421
22.2.1	struct gpio_pin_config_t	421
22.3	Macro Definition Documentation	422
22.3.1	FSL_GPIO_DRIVER_VERSION	422
22.4	Enumeration Type Documentation	422
22.4.1	gpio_pin_direction_t	422
22.5	GPIO Driver	423
22.5.1	Overview	423
22.5.2	Typical use case	423
22.5.3	Function Documentation	424
22.6	FGPIO Driver	427
22.6.1	Typical use case	427

Chapter I2C: Inter-Integrated Circuit Driver

23.1	Overview	429
23.2	I2C Driver	430
23.2.1	Overview	430
23.2.2	Typical use case	430

Contents

Section Number	Title	Page Number
23.2.3	Data Structure Documentation	435
23.2.4	Macro Definition Documentation	439
23.2.5	Typedef Documentation	439
23.2.6	Enumeration Type Documentation	439
23.2.7	Function Documentation	441
23.3	I2C eDMA Driver	455
23.3.1	Overview	455
23.3.2	Data Structure Documentation	455
23.3.3	Macro Definition Documentation	457
23.3.4	Typedef Documentation	457
23.3.5	Function Documentation	457
23.4	I2C FreeRTOS Driver	460
23.4.1	Overview	460
23.4.2	Macro Definition Documentation	460
23.4.3	Function Documentation	460
23.5	I2C CMSIS Driver	463
23.5.1	I2C CMSIS Driver	463
Chapter	LLWU: Low-Leakage Wakeup Unit Driver	
24.1	Overview	465
24.2	External wakeup pins configurations	465
24.3	Internal wakeup modules configurations	465
24.4	Digital pin filter for external wakeup pin configurations	465
24.5	Data Structure Documentation	466
24.5.1	struct llwu_external_pin_filter_mode_t	466
24.6	Macro Definition Documentation	466
24.6.1	FSL_LLWU_DRIVER_VERSION	466
24.7	Enumeration Type Documentation	467
24.7.1	llwu_external_pin_mode_t	467
24.7.2	llwu_pin_filter_mode_t	467
24.8	Function Documentation	467
24.8.1	LLWU_SetExternalWakeupPinMode	467
24.8.2	LLWU_GetExternalWakeupPinFlag	467
24.8.3	LLWU_ClearExternalWakeupPinFlag	468
24.8.4	LLWU_EnableInternalModuleInterruptWakup	468

Contents

Section Number	Title	Page Number
24.8.5	LLWU_GetInternalWakeupModuleFlag	468
24.8.6	LLWU_SetPinFilterMode	469
24.8.7	LLWU_GetPinFilterFlag	469
24.8.8	LLWU_ClearPinFilterFlag	469
24.8.9	LLWU_SetResetPinMode	470
Chapter	LPTMR: Low-Power Timer	
25.1	Overview	471
25.2	Function groups	471
25.2.1	Initialization and deinitialization	471
25.2.2	Timer period Operations	471
25.2.3	Start and Stop timer operations	471
25.2.4	Status	472
25.2.5	Interrupt	472
25.3	Typical use case	472
25.3.1	LPTMR tick example	472
25.4	Data Structure Documentation	474
25.4.1	struct lptmr_config_t	474
25.5	Enumeration Type Documentation	475
25.5.1	lptmr_pin_select_t	475
25.5.2	lptmr_pin_polarity_t	475
25.5.3	lptmr_timer_mode_t	475
25.5.4	lptmr_prescaler_glitch_value_t	475
25.5.5	lptmr_prescaler_clock_select_t	476
25.5.6	lptmr_interrupt_enable_t	476
25.5.7	lptmr_status_flags_t	476
25.6	Function Documentation	476
25.6.1	LPTMR_Init	476
25.6.2	LPTMR_Deinit	477
25.6.3	LPTMR_GetDefaultConfig	477
25.6.4	LPTMR_EnableInterrupts	477
25.6.5	LPTMR_DisableInterrupts	477
25.6.6	LPTMR_GetEnabledInterrupts	478
25.6.7	LPTMR_GetStatusFlags	478
25.6.8	LPTMR_ClearStatusFlags	478
25.6.9	LPTMR_SetTimerPeriod	479
25.6.10	LPTMR_GetCurrentTimerCount	479
25.6.11	LPTMR_StartTimer	479
25.6.12	LPTMR_StopTimer	480

Contents

Section Number	Title	Page Number
Chapter	PDB: Programmable Delay Block	
26.1	Overview	481
26.2	Typical use case	481
26.2.1	Working as basic PDB counter with a PDB interrupt.	481
26.2.2	Working with an additional trigger. The ADC trigger is used as an example.	481
26.3	Data Structure Documentation	485
26.3.1	struct pdb_config_t	485
26.3.2	struct pdb_adc_pretrigger_config_t	486
26.3.3	struct pdb_dac_trigger_config_t	486
26.4	Macro Definition Documentation	487
26.4.1	FSL_PDB_DRIVER_VERSION	487
26.5	Enumeration Type Documentation	487
26.5.1	_pdb_status_flags	487
26.5.2	_pdb_adc_pretrigger_flags	487
26.5.3	_pdb_interrupt_enable	487
26.5.4	pdb_load_value_mode_t	487
26.5.5	pdb_prescaler_divider_t	488
26.5.6	pdb_divider_multiplication_factor_t	488
26.5.7	pdb_trigger_input_source_t	488
26.5.8	pdb_adc_trigger_channel_t	489
26.5.9	pdb_adc_pretrigger_t	489
26.5.10	pdb_dac_trigger_channel_t	490
26.5.11	pdb_pulse_out_trigger_channel_t	490
26.5.12	pdb_pulse_out_channel_mask_t	490
26.6	Function Documentation	491
26.6.1	PDB_Init	491
26.6.2	PDB_Deinit	491
26.6.3	PDB_GetDefaultConfig	491
26.6.4	PDB_Enable	491
26.6.5	PDB_DoSoftwareTrigger	492
26.6.6	PDB_DoLoadValues	492
26.6.7	PDB_EnableDMA	492
26.6.8	PDB_EnableInterrupts	492
26.6.9	PDB_DisableInterrupts	493
26.6.10	PDB_GetStatusFlags	493
26.6.11	PDB_ClearStatusFlags	493
26.6.12	PDB_SetModulusValue	493
26.6.13	PDB_GetCounterValue	494
26.6.14	PDB_SetCounterDelayValue	494
26.6.15	PDB_SetADCPreTriggerConfig	494

Contents

Section Number	Title	Page Number
26.6.16	PDB_SetADCPreTriggerDelayValue	495
26.6.17	PDB_GetADCPreTriggerStatusFlags	495
26.6.18	PDB_ClearADCPreTriggerStatusFlags	495
26.6.19	PDB_SetDACTriggerConfig	496
26.6.20	PDB_SetDACTriggerIntervalValue	496
26.6.21	PDB_EnablePulseOutTrigger	496
26.6.22	PDB_SetPulseOutTriggerDelayValue	497

Chapter PIT: Periodic Interrupt Timer

27.1	Overview	499
27.2	Function groups	499
27.2.1	Initialization and deinitialization	499
27.2.2	Timer period Operations	499
27.2.3	Start and Stop timer operations	499
27.2.4	Status	500
27.2.5	Interrupt	500
27.3	Typical use case	500
27.3.1	PIT tick example	500
27.4	Data Structure Documentation	501
27.4.1	struct pit_config_t	501
27.5	Enumeration Type Documentation	501
27.5.1	pit_chnl_t	501
27.5.2	pit_interrupt_enable_t	502
27.5.3	pit_status_flags_t	502
27.6	Function Documentation	502
27.6.1	PIT_Init	502
27.6.2	PIT_Deinit	502
27.6.3	PIT_GetDefaultConfig	503
27.6.4	PIT_SetTimerChainMode	503
27.6.5	PIT_EnableInterrupts	503
27.6.6	PIT_DisableInterrupts	504
27.6.7	PIT_GetEnabledInterrupts	504
27.6.8	PIT_GetStatusFlags	504
27.6.9	PIT_ClearStatusFlags	505
27.6.10	PIT_SetTimerPeriod	505
27.6.11	PIT_GetCurrentTimerCount	506
27.6.12	PIT_StartTimer	506
27.6.13	PIT_StopTimer	506

Contents

Section Number	Title	Page Number
Chapter	PMC: Power Management Controller	
28.1	Overview	509
28.2	Data Structure Documentation	510
28.2.1	struct pmc_low_volt_detect_config_t	510
28.2.2	struct pmc_low_volt_warning_config_t	510
28.2.3	struct pmc_bandgap_buffer_config_t	510
28.3	Macro Definition Documentation	511
28.3.1	FSL_PMC_DRIVER_VERSION	511
28.4	Enumeration Type Documentation	511
28.4.1	pmc_low_volt_detect_volt_select_t	511
28.4.2	pmc_low_volt_warning_volt_select_t	511
28.5	Function Documentation	511
28.5.1	PMC_ConfigureLowVoltDetect	511
28.5.2	PMC_GetLowVoltDetectFlag	512
28.5.3	PMC_ClearLowVoltDetectFlag	512
28.5.4	PMC_ConfigureLowVoltWarning	512
28.5.5	PMC_GetLowVoltWarningFlag	513
28.5.6	PMC_ClearLowVoltWarningFlag	513
28.5.7	PMC_ConfigureBandgapBuffer	513
28.5.8	PMC_GetPeriphIOIsolationFlag	514
28.5.9	PMC_ClearPeriphIOIsolationFlag	514
28.5.10	PMC_IsRegulatorInRunRegulation	514
Chapter	PORT: Port Control and Interrupts	
29.1	Overview	517
29.2	Data Structure Documentation	519
29.2.1	struct port_digital_filter_config_t	519
29.2.2	struct port_pin_config_t	519
29.3	Macro Definition Documentation	520
29.3.1	FSL_PORT_DRIVER_VERSION	520
29.4	Enumeration Type Documentation	520
29.4.1	_port_pull	520
29.4.2	_port_slew_rate	520
29.4.3	_port_open_drain_enable	520
29.4.4	_port_passive_filter_enable	520
29.4.5	_port_drive_strength	521
29.4.6	_port_lock_register	521

Contents

Section Number	Title	Page Number
29.4.7	port_mux_t	521
29.4.8	port_interrupt_t	521
29.4.9	port_digital_filter_clock_source_t	522
29.5	Function Documentation	522
29.5.1	PORT_SetPinConfig	522
29.5.2	PORT_SetMultiplePinsConfig	523
29.5.3	PORT_SetPinMux	523
29.5.4	PORT_EnablePinsDigitalFilter	524
29.5.5	PORT_SetDigitalFilterConfig	524
29.5.6	PORT_SetPinInterruptConfig	525
29.5.7	PORT_SetPinDriveStrength	525
29.5.8	PORT_GetPinsInterruptFlags	526
29.5.9	PORT_ClearPinsInterruptFlags	526
Chapter	RCM: Reset Control Module Driver	
30.1	Overview	527
30.2	Data Structure Documentation	528
30.2.1	struct rcm_reset_pin_filter_config_t	528
30.3	Macro Definition Documentation	528
30.3.1	FSL_RCM_DRIVER_VERSION	528
30.4	Enumeration Type Documentation	528
30.4.1	rcm_reset_source_t	528
30.4.2	rcm_run_wait_filter_mode_t	529
30.5	Function Documentation	529
30.5.1	RCM_GetPreviousResetSources	529
30.5.2	RCM_ConfigureResetPinFilter	529
30.5.3	RCM_GetEasyPortModePinStatus	530
Chapter	RNGA: Random Number Generator Accelerator Driver	
31.1	Overview	531
31.2	RNGA Initialization	531
31.3	Get random data from RNGA	531
31.4	RNGA Set/Get Working Mode	531
31.5	Seed RNGA	531

Contents

Section Number	Title	Page Number
31.6	Macro Definition Documentation	532
31.6.1	FSL_RNGA_DRIVER_VERSION	532
31.7	Enumeration Type Documentation	533
31.7.1	rnga_mode_t	533
31.8	Function Documentation	533
31.8.1	RNGA_Init	533
31.8.2	RNGA_Deinit	533
31.8.3	RNGA_GetRandomData	533
31.8.4	RNGA_Seed	534
31.8.5	RNGA_SetMode	534
31.8.6	RNGA_GetMode	534
Chapter	RTC: Real Time Clock	
32.1	Overview	535
32.2	Function groups	535
32.2.1	Initialization and deinitialization	535
32.2.2	Set & Get Datetime	535
32.2.3	Set & Get Alarm	535
32.2.4	Start & Stop timer	535
32.2.5	Status	536
32.2.6	Interrupt	536
32.2.7	RTC Oscillator	536
32.2.8	Monotonic Counter	536
32.3	Typical use case	536
32.3.1	RTC tick example	536
32.4	Data Structure Documentation	538
32.4.1	struct rtc_datetime_t	538
32.4.2	struct rtc_config_t	539
32.5	Enumeration Type Documentation	539
32.5.1	rtc_interrupt_enable_t	539
32.5.2	rtc_status_flags_t	540
32.5.3	rtc_osc_cap_load_t	540
32.6	Function Documentation	540
32.6.1	RTC_Init	540
32.6.2	RTC_Deinit	540
32.6.3	RTC_GetDefaultConfig	541
32.6.4	RTC_SetDatetime	541
32.6.5	RTC_GetDatetime	541

Contents

Section Number	Title	Page Number
32.6.6	RTC_SetAlarm	542
32.6.7	RTC_GetAlarm	542
32.6.8	RTC_EnableInterrupts	542
32.6.9	RTC_DisableInterrupts	543
32.6.10	RTC_GetEnabledInterrupts	543
32.6.11	RTC_GetStatusFlags	543
32.6.12	RTC_ClearStatusFlags	543
32.6.13	RTC_SetClockSource	544
32.6.14	RTC_StartTimer	544
32.6.15	RTC_StopTimer	544
32.6.16	RTC_SetOscCapLoad	544
32.6.17	RTC_Reset	545
32.6.18	RTC_EnableWakeUpPin	545
Chapter	SAI: Serial Audio Interface	
33.1	Overview	547
33.2	Typical configurations	547
33.3	Typical use case	548
33.3.1	SAI Send/receive using an interrupt method	548
33.3.2	SAI Send/receive using a DMA method	548
33.4	SAI Driver	549
33.4.1	Overview	549
33.4.2	Data Structure Documentation	556
33.4.3	Macro Definition Documentation	561
33.4.4	Enumeration Type Documentation	561
33.4.5	Function Documentation	565
33.5	SAI EDMA Driver	593
33.5.1	Overview	593
33.5.2	Data Structure Documentation	594
33.5.3	Function Documentation	595
Chapter	SDHC: Secure Digital Host Controller Driver	
34.1	Overview	603
34.2	Typical use case	603
34.2.1	SDHC Operation	603
34.3	Data Structure Documentation	611
34.3.1	struct sdhc_adma2_descriptor_t	611
34.3.2	struct sdhc_capability_t	611

Contents

Section Number	Title	Page Number
34.3.3	struct sdhc_transfer_config_t	612
34.3.4	struct sdhc_boot_config_t	612
34.3.5	struct sdhc_config_t	613
34.3.6	struct sdhc_data_t	613
34.3.7	struct sdhc_command_t	614
34.3.8	struct sdhc_transfer_t	614
34.3.9	struct sdhc_transfer_callback_t	614
34.3.10	struct _sdhc_handle	615
34.3.11	struct sdhc_host_t	615
34.4	Macro Definition Documentation	615
34.4.1	FSL_SDHC_DRIVER_VERSION	615
34.5	Typedef Documentation	616
34.5.1	sdhc_adma1_descriptor_t	616
34.5.2	sdhc_transfer_function_t	616
34.6	Enumeration Type Documentation	616
34.6.1	anonymous enum	616
34.6.2	anonymous enum	616
34.6.3	anonymous enum	616
34.6.4	anonymous enum	617
34.6.5	anonymous enum	617
34.6.6	anonymous enum	617
34.6.7	anonymous enum	618
34.6.8	anonymous enum	618
34.6.9	anonymous enum	619
34.6.10	sdhc_adma_error_state_t	619
34.6.11	anonymous enum	619
34.6.12	sdhc_data_bus_width_t	620
34.6.13	sdhc_endian_mode_t	620
34.6.14	sdhc_dma_mode_t	620
34.6.15	anonymous enum	620
34.6.16	sdhc_boot_mode_t	620
34.6.17	sdhc_card_command_type_t	621
34.6.18	sdhc_card_response_type_t	621
34.6.19	anonymous enum	621
34.6.20	anonymous enum	622
34.7	Function Documentation	622
34.7.1	SDHC_Init	622
34.7.2	SDHC_Deinit	622
34.7.3	SDHC_Reset	623
34.7.4	SDHC_SetAdmaTableConfig	623
34.7.5	SDHC_EnableInterruptStatus	624

Contents

Section Number	Title	Page Number
34.7.6	SDHC_DisableInterruptStatus	624
34.7.7	SDHC_EnableInterruptSignal	624
34.7.8	SDHC_DisableInterruptSignal	624
34.7.9	SDHC_GetEnabledInterruptStatusFlags	625
34.7.10	SDHC_GetInterruptStatusFlags	625
34.7.11	SDHC_ClearInterruptStatusFlags	625
34.7.12	SDHC_GetAutoCommand12ErrorStatusFlags	625
34.7.13	SDHC_GetAdmaErrorStatusFlags	626
34.7.14	SDHC_GetPresentStatusFlags	626
34.7.15	SDHC_GetCapability	626
34.7.16	SDHC_EnableSdClock	627
34.7.17	SDHC_SetSdClock	627
34.7.18	SDHC_SetCardActive	627
34.7.19	SDHC_SetDataBusWidth	628
34.7.20	SDHC_CardDetectByData3	628
34.7.21	SDHC_SetTransferConfig	628
34.7.22	SDHC_GetCommandResponse	629
34.7.23	SDHC_WriteData	629
34.7.24	SDHC_ReadData	629
34.7.25	SDHC_EnableWakeupEvent	630
34.7.26	SDHC_EnableCardDetectTest	630
34.7.27	SDHC_SetCardDetectTestLevel	630
34.7.28	SDHC_EnableSdioControl	631
34.7.29	SDHC_SetContinueRequest	631
34.7.30	SDHC_SetMmcBootConfig	631
34.7.31	SDHC_SetForceEvent	632
34.7.32	SDHC_TransferBlocking	632
34.7.33	SDHC_TransferCreateHandle	633
34.7.34	SDHC_TransferNonBlocking	633
34.7.35	SDHC_TransferHandleIRQ	634

Chapter SIM: System Integration Module Driver

35.1	Overview	635
35.2	Data Structure Documentation	635
35.2.1	struct sim_uid_t	635
35.3	Enumeration Type Documentation	636
35.3.1	_sim_usb_volt_reg_enable_mode	636
35.3.2	_sim_flash_mode	636
35.4	Function Documentation	636
35.4.1	SIM_SetUsbVoltRegulatorEnableMode	636
35.4.2	SIM_GetUniqueId	637

Contents

Section Number	Title	Page Number
35.4.3	SIM_SetFlashMode	637
Chapter	SMC: System Mode Controller Driver	
36.1	Overview	639
36.2	Typical use case	639
36.2.1	Enter wait or stop modes	639
36.3	Data Structure Documentation	641
36.3.1	struct smc_power_mode_vlls_config_t	641
36.4	Enumeration Type Documentation	642
36.4.1	smc_power_mode_protection_t	642
36.4.2	smc_power_state_t	642
36.4.3	smc_run_mode_t	642
36.4.4	smc_stop_mode_t	642
36.4.5	smc_stop_submode_t	643
36.4.6	smc_partial_stop_option_t	643
36.4.7	anonymous enum	643
36.5	Function Documentation	643
36.5.1	SMC_SetPowerModeProtection	643
36.5.2	SMC_GetPowerModeState	644
36.5.3	SMC_PreEnterStopModes	644
36.5.4	SMC_PostExitStopModes	644
36.5.5	SMC_PreEnterWaitModes	644
36.5.6	SMC_PostExitWaitModes	644
36.5.7	SMC_SetPowerModeRun	645
36.5.8	SMC_SetPowerModeWait	646
36.5.9	SMC_SetPowerModeStop	646
36.5.10	SMC_SetPowerModeVlpr	646
36.5.11	SMC_SetPowerModeVlpw	647
36.5.12	SMC_SetPowerModeVlps	647
36.5.13	SMC_SetPowerModeLls	647
36.5.14	SMC_SetPowerModeVlls	647
Chapter	SYSMPU: System Memory Protection Unit	
37.1	Overview	649
37.2	Initialization and Deinitialization	649
37.3	Basic Control Operations	649
37.4	Data Structure Documentation	652

Contents

Section Number	Title	Page Number
37.4.1	struct sysmpu_hardware_info_t	652
37.4.2	struct sysmpu_access_err_info_t	653
37.4.3	struct sysmpu_rwxrights_master_access_control_t	653
37.4.4	struct sysmpu_rwrights_master_access_control_t	654
37.4.5	struct sysmpu_region_config_t	654
37.4.6	struct sysmpu_config_t	655
37.5	Macro Definition Documentation	656
37.5.1	FSL_SYSPU_DRIVER_VERSION	656
37.5.2	SYSMPU_MASTER_RWATTRIBUTE_START_PORT	656
37.5.3	SYSMPU_REGION_RWXRIGHTS_MASTER_SHIFT	656
37.5.4	SYSMPU_REGION_RWXRIGHTS_MASTER_MASK	656
37.5.5	SYSMPU_REGION_RWXRIGHTS_MASTER_WIDTH	656
37.5.6	SYSMPU_REGION_RWXRIGHTS_MASTER	656
37.5.7	SYSMPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT	656
37.5.8	SYSMPU_REGION_RWXRIGHTS_MASTER_PE_MASK	656
37.5.9	SYSMPU_REGION_RWXRIGHTS_MASTER_PE	656
37.5.10	SYSMPU_REGION_RWRIGHTS_MASTER_SHIFT	657
37.5.11	SYSMPU_REGION_RWRIGHTS_MASTER_MASK	657
37.5.12	SYSMPU_REGION_RWRIGHTS_MASTER	657
37.6	Enumeration Type Documentation	657
37.6.1	sysmpu_region_total_num_t	657
37.6.2	sysmpu_slave_t	657
37.6.3	sysmpu_err_access_control_t	657
37.6.4	sysmpu_err_access_type_t	658
37.6.5	sysmpu_err_attributes_t	658
37.6.6	sysmpu_supervisor_access_rights_t	658
37.6.7	sysmpu_user_access_rights_t	658
37.7	Function Documentation	659
37.7.1	SYSMPU_Init	659
37.7.2	SYSMPU_Deinit	660
37.7.3	SYSMPU_Enable	660
37.7.4	SYSMPU_RegionEnable	660
37.7.5	SYSMPU_GetHardwareInfo	660
37.7.6	SYSMPU_SetRegionConfig	661
37.7.7	SYSMPU_SetRegionAddr	661
37.7.8	SYSMPU_SetRegionRwxMasterAccessRights	662
37.7.9	SYSMPU_SetRegionRwMasterAccessRights	662
37.7.10	SYSMPU_GetSlavePortErrorStatus	663
37.7.11	SYSMPU_GetDetailErrorAccessInfo	663

Contents

Section Number	Title	Page Number
Chapter	UART: Universal Asynchronous Receiver/Transmitter Driver	
38.1	Overview	665
38.2	UART Driver	666
38.2.1	Overview	666
38.2.2	Typical use case	666
38.2.3	Data Structure Documentation	671
38.2.4	Macro Definition Documentation	674
38.2.5	Typedef Documentation	674
38.2.6	Enumeration Type Documentation	674
38.2.7	Function Documentation	677
38.3	UART eDMA Driver	693
38.3.1	Overview	693
38.3.2	Data Structure Documentation	694
38.3.3	Macro Definition Documentation	695
38.3.4	Typedef Documentation	695
38.3.5	Function Documentation	695
38.4	UART FreeRTOS Driver	699
38.4.1	Overview	699
38.4.2	Data Structure Documentation	699
38.4.3	Macro Definition Documentation	700
38.4.4	Function Documentation	700
38.5	UART CMSIS Driver	702
38.5.1	UART CMSIS Driver	702
Chapter	VREF: Voltage Reference Driver	
39.1	Overview	705
39.2	VREF functional Operation	705
39.3	Typical use case and example	705
39.4	Data Structure Documentation	706
39.4.1	struct vref_config_t	706
39.5	Macro Definition Documentation	706
39.5.1	FSL_VREF_DRIVER_VERSION	706
39.6	Enumeration Type Documentation	706
39.6.1	vref_buffer_mode_t	706
39.7	Function Documentation	706

Contents

Section Number	Title	Page Number
39.7.1	VREF_Init	706
39.7.2	VREF_Deinit	707
39.7.3	VREF_GetDefaultConfig	707
39.7.4	VREF_SetTrimVal	707
39.7.5	VREF_GetTrimVal	708
Chapter	WDOG: Watchdog Timer Driver	
40.1	Overview	709
40.2	Typical use case	709
40.3	Data Structure Documentation	711
40.3.1	struct wdog_work_mode_t	711
40.3.2	struct wdog_config_t	711
40.3.3	struct wdog_test_config_t	712
40.4	Macro Definition Documentation	712
40.4.1	FSL_WDOG_DRIVER_VERSION	712
40.5	Enumeration Type Documentation	712
40.5.1	wdog_clock_source_t	712
40.5.2	wdog_clock_prescaler_t	712
40.5.3	wdog_test_mode_t	713
40.5.4	wdog_tested_byte_t	713
40.5.5	_wdog_interrupt_enable_t	713
40.5.6	_wdog_status_flags_t	713
40.6	Function Documentation	713
40.6.1	WDOG_GetDefaultConfig	713
40.6.2	WDOG_Init	714
40.6.3	WDOG_Deinit	714
40.6.4	WDOG_SetTestModeConfig	715
40.6.5	WDOG_Enable	715
40.6.6	WDOG_Disable	715
40.6.7	WDOG_EnableInterrupts	716
40.6.8	WDOG_DisableInterrupts	716
40.6.9	WDOG_GetStatusFlags	716
40.6.10	WDOG_ClearStatusFlags	717
40.6.11	WDOG_SetTimeoutValue	717
40.6.12	WDOG_SetWindowValue	718
40.6.13	WDOG_Unlock	718
40.6.14	WDOG_Refresh	718
40.6.15	WDOG_GetResetCount	719
40.6.16	WDOG_ClearResetCount	720

Contents

Section Number	Title	Page Number
Chapter	Debug Console	
41.1	Overview	721
41.2	Function groups	721
41.2.1	Initialization	721
41.2.2	Advanced Feature	722
41.3	Typical use case	726
41.4	Macro Definition Documentation	728
41.4.1	DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN	728
41.4.2	DEBUGCONSOLE_REDIRECT_TO_SDK	728
41.4.3	DEBUGCONSOLE_DISABLE	728
41.4.4	SDK_DEBUGCONSOLE	728
41.4.5	PRINTF	728
41.5	Function Documentation	729
41.5.1	DbgConsole_Init	729
41.5.2	DbgConsole_Deinit	729
41.5.3	DbgConsole_Printf	729
41.5.4	DbgConsole_Putchar	730
41.5.5	DbgConsole_Scanf	730
41.5.6	DbgConsole_Getchar	731
41.5.7	DbgConsole_BlockingPrintf	731
41.5.8	DbgConsole_Flush	731
41.5.9	StrFormatPrintf	732
41.5.10	StrFormatScanf	732
41.6	Semihosting	733
41.6.1	Guide Semihosting for IAR	733
41.6.2	Guide Semihosting for Keil μ Vision	733
41.6.3	Guide Semihosting for MCUXpresso IDE	734
41.6.4	Guide Semihosting for ARMGCC	734
41.7	SWO	737
41.7.1	Guide SWO for SDK	737
41.7.2	Guide SWO for Keil μ Vision	738
41.7.3	Guide SWO for MCUXpresso IDE	739
41.7.4	Guide SWO for ARMGCC	739
Chapter	Notification Framework	
42.1	Overview	741
42.2	Notifier Overview	741

Contents

Section Number	Title	Page Number
42.3	Data Structure Documentation	743
42.3.1	struct notifier_notification_block_t	743
42.3.2	struct notifier_callback_config_t	744
42.3.3	struct notifier_handle_t	744
42.4	Typedef Documentation	745
42.4.1	notifier_user_config_t	745
42.4.2	notifier_user_function_t	745
42.4.3	notifier_callback_t	746
42.5	Enumeration Type Documentation	746
42.5.1	_notifier_status	746
42.5.2	notifier_policy_t	747
42.5.3	notifier_notification_type_t	747
42.5.4	notifier_callback_type_t	747
42.6	Function Documentation	748
42.6.1	NOTIFIER_CreateHandle	748
42.6.2	NOTIFIER_SwitchConfig	749
42.6.3	NOTIFIER_GetErrorCallbackIndex	750
Chapter	Shell	
43.1	Overview	751
43.2	Function groups	751
43.2.1	Initialization	751
43.2.2	Advanced Feature	751
43.2.3	Shell Operation	751
43.3	Data Structure Documentation	753
43.3.1	struct shell_command_t	753
43.4	Macro Definition Documentation	754
43.4.1	SHELL_NON_BLOCKING_MODE	754
43.4.2	SHELL_AUTO_COMPLETE	754
43.4.3	SHELL_BUFFER_SIZE	754
43.4.4	SHELL_MAX_ARGS	754
43.4.5	SHELL_HISTORY_COUNT	754
43.4.6	SHELL_HANDLE_SIZE	754
43.4.7	SHELL_USE_COMMON_TASK	754
43.4.8	SHELL_TASK_PRIORITY	754
43.4.9	SHELL_TASK_STACK_SIZE	754
43.4.10	SHELL_HANDLE_DEFINE	754
43.4.11	SHELL_COMMAND_DEFINE	755
43.4.12	SHELL_COMMAND	755

Contents

Section Number	Title	Page Number
43.5	Typedef Documentation	756
43.5.1	cmd_function_t	756
43.6	Enumeration Type Documentation	756
43.6.1	shell_status_t	756
43.7	Function Documentation	756
43.7.1	SHELL_Init	756
43.7.2	SHELL_RegisterCommand	757
43.7.3	SHELL_UnregisterCommand	757
43.7.4	SHELL_Write	758
43.7.5	SHELL_Printf	758
43.7.6	SHELL_Task	759
Chapter	Secure Digital Card/Embedded MultiMedia Card/SDIO card	
44.1	Overview	761
44.2	SDIO Card Driver	762
44.2.1	Overview	762
44.2.2	SDIO CARD Operation	762
44.2.3	Data Structure Documentation	764
44.2.4	Macro Definition Documentation	766
44.2.5	Enumeration Type Documentation	766
44.2.6	Function Documentation	766
44.3	SD Card Driver	785
44.3.1	Overview	785
44.3.2	SD CARD Operation	785
44.3.3	Data Structure Documentation	788
44.3.4	Macro Definition Documentation	789
44.3.5	Enumeration Type Documentation	789
44.3.6	Function Documentation	789
44.4	MMC Card Driver	799
44.4.1	Overview	799
44.4.2	MMC CARD Operation	799
44.4.3	Data Structure Documentation	801
44.4.4	Macro Definition Documentation	803
44.4.5	Enumeration Type Documentation	803
44.4.6	Function Documentation	803
44.5	HOST Driver	813
44.5.1	Overview	813
44.6	SDMMC OSA	814

Contents

Section Number	Title	Page Number
44.6.1	Overview	814
44.6.2	Function Documentation	814
44.6.3	SDHC HOST adapter Driver	817
44.7	SDMMC Common	827
44.7.1	Overview	827
44.7.2	Data Structure Documentation	846
44.7.3	Macro Definition Documentation	859
44.7.4	Enumeration Type Documentation	859
44.7.5	Function Documentation	876
Chapter	SPI based Secure Digital Card (SDSPI)	
45.1	Overview	881
45.2	Data Structure Documentation	883
45.2.1	struct sdspi_host_t	883
45.2.2	struct sdspi_card_t	883
45.3	Macro Definition Documentation	884
45.3.1	FSL_SDSPI_DRIVER_VERSION	884
45.3.2	DSPI_DUMMY_DATA	884
45.3.3	SDSPI_CARD_CRC_PROTECTION_ENABLE	884
45.4	Enumeration Type Documentation	885
45.4.1	_sdspi_status	885
45.4.2	_sdspi_card_flag	885
45.4.3	_sdspi_response_type	885
45.4.4	_sdspi_cmd	886
45.5	Function Documentation	886
45.5.1	SDSPI_Init	886
45.5.2	SDSPI_Deinit	887
45.5.3	SDSPI_CheckReadOnly	887
45.5.4	SDSPI_ReadBlocks	887
45.5.5	SDSPI_WriteBlocks	888
45.5.6	SDSPI_SendCid	889
45.5.7	SDSPI_SendPreErase	889
45.5.8	SDSPI_EraseBlocks	890
45.5.9	SDSPI_SwitchToHighSpeed	890
Chapter	CODEC codec Driver	
46.1	Overview	893
46.2	codec common Driver	894

Contents

Section Number	Title	Page Number
46.2.1	Overview	894
46.2.2	Data Structure Documentation	898
46.2.3	Macro Definition Documentation	899
46.2.4	Enumeration Type Documentation	899
46.2.5	Function Documentation	904
46.3	cs42888 Driver	908
46.3.1	Overview	908
46.3.2	Data Structure Documentation	910
46.3.3	Macro Definition Documentation	911
46.3.4	Enumeration Type Documentation	911
46.3.5	Function Documentation	912
46.3.6	cs42888 adapter	918
46.4	da7212 Driver	924
46.4.1	Overview	924
46.4.2	Data Structure Documentation	927
46.4.3	Macro Definition Documentation	928
46.4.4	Enumeration Type Documentation	928
46.4.5	Function Documentation	930
46.4.6	da7212 adapter	935
46.5	sgtl5000 Driver	941
46.5.1	Overview	941
46.5.2	Data Structure Documentation	943
46.5.3	Macro Definition Documentation	945
46.5.4	Enumeration Type Documentation	945
46.5.5	Function Documentation	946
46.5.6	sgtl5000 adapter	953
46.6	wm8960 Driver	959
46.6.1	Overview	959
46.6.2	Data Structure Documentation	962
46.6.3	Macro Definition Documentation	964
46.6.4	Enumeration Type Documentation	964
46.6.5	Function Documentation	966
46.6.6	wm8960 adapter	972
46.7	wm8904 Driver	978
46.7.1	Overview	978
46.7.2	Data Structure Documentation	982
46.7.3	Macro Definition Documentation	983
46.7.4	Enumeration Type Documentation	983
46.7.5	Function Documentation	986
46.7.6	wm8904 adapter	995

Contents

Section Number Chapter	Title	Page Number
	Serial_Manager	
47.1	Overview	1001
47.2	Data Structure Documentation	1003
47.2.1	struct serial_manager_config_t	1003
47.2.2	struct serial_manager_callback_message_t	1003
47.3	Macro Definition Documentation	1004
47.3.1	SERIAL_MANAGER_HANDLE_SIZE	1004
47.3.2	SERIAL_MANAGER_HANDLE_DEFINE	1004
47.3.3	SERIAL_MANAGER_WRITE_HANDLE_DEFINE	1004
47.3.4	SERIAL_MANAGER_READ_HANDLE_DEFINE	1005
47.3.5	SERIAL_MANAGER_USE_COMMON_TASK	1005
47.3.6	SERIAL_MANAGER_TASK_PRIORITY	1005
47.3.7	SERIAL_MANAGER_TASK_STACK_SIZE	1005
47.4	Enumeration Type Documentation	1005
47.4.1	serial_port_type_t	1005
47.4.2	serial_manager_status_t	1006
47.5	Function Documentation	1006
47.5.1	SerialManager_Init	1006
47.5.2	SerialManager_Deinit	1007
47.5.3	SerialManager_OpenWriteHandle	1008
47.5.4	SerialManager_CloseWriteHandle	1009
47.5.5	SerialManager_OpenReadHandle	1009
47.5.6	SerialManager_CloseReadHandle	1010
47.5.7	SerialManager_WriteBlocking	1010
47.5.8	SerialManager_ReadBlocking	1011
47.5.9	SerialManager_EnterLowpower	1012
47.5.10	SerialManager_ExitLowpower	1012
47.6	Serial Port Uart	1014
47.6.1	Overview	1014
47.6.2	Data Structure Documentation	1014
47.6.3	Enumeration Type Documentation	1015
47.7	Serial Port USB	1016
47.7.1	Overview	1016
47.7.2	Data Structure Documentation	1017
47.7.3	Enumeration Type Documentation	1017
47.7.4	USB Device Configuration	1018
47.8	Serial Port SWO	1019
47.8.1	Overview	1019

Contents

Section Number	Title	Page Number
47.8.2	Data Structure Documentation1019
47.8.3	Enumeration Type Documentation1019
Chapter	Data Structure Documentation	
48.0.4	codec_i2c_config_t Struct Reference1021

Chapter 1

Introduction

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

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

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

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

- IAR Embedded Workbench
- GNU Arm Embedded Toolchain

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

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

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

Table 2: MCUXpresso SDK Folder Structure

Chapter 2

Trademarks

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

How to Reach Us:

Home Page: nxp.com

Web Support: nxp.com/support

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

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

© 2019 NXP B.V.



Chapter 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

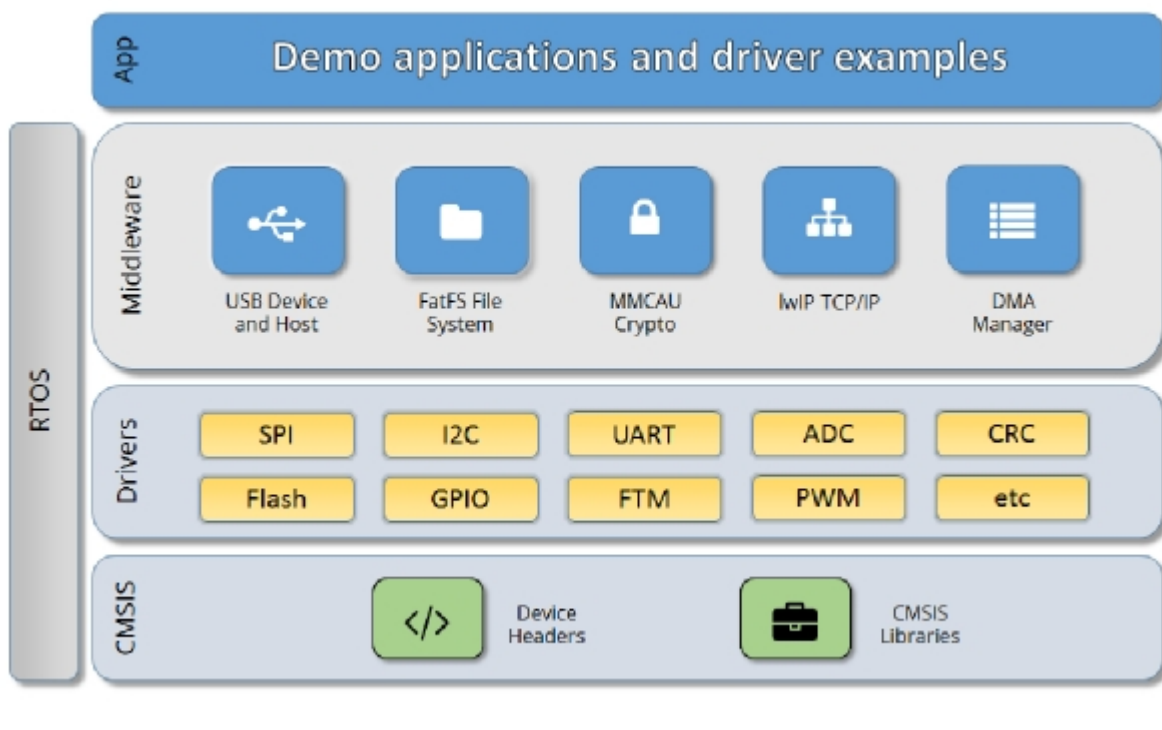


Figure 1: MCUXpresso SDK Block Diagram

MCU header files

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

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

CMSIS Support

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

MCUXpresso SDK Peripheral Drivers

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

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

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

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

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

Interrupt handling for transactional APIs

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

```
PUBWEAK SPI0_IRQHandler
PUBWEAK SPI0_DriverIRQHandler
SPI0_IRQHandler
```



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

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

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

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

Feature Header Files

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

Application

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



Chapter 4

Driver errors status

- [kStatus_DSPI_Error](#) = 601
- [kStatus_EDMA_QueueFull](#) = 5100
- [kStatus_EDMA_Busy](#) = 5101
- [kStatus_ENET_RxFrameError](#) = 4000
- [kStatus_ENET_RxFrameFail](#) = 4001
- [kStatus_ENET_RxFrameEmpty](#) = 4002
- [kStatus_ENET_TxFrameOverLen](#) = 4003
- [kStatus_ENET_TxFrameBusy](#) = 4004
- [kStatus_ENET_TxFrameFail](#) = 4005
- [kStatus_FLEXCAN_TxBusy](#) = 5300
- [kStatus_FLEXCAN_TxIdle](#) = 5301
- [kStatus_FLEXCAN_TxSwitchToRx](#) = 5302
- [kStatus_FLEXCAN_RxBusy](#) = 5303
- [kStatus_FLEXCAN_RxIdle](#) = 5304
- [kStatus_FLEXCAN_RxOverflow](#) = 5305
- [kStatus_FLEXCAN_RxFifoBusy](#) = 5306
- [kStatus_FLEXCAN_RxFifoIdle](#) = 5307
- [kStatus_FLEXCAN_RxFifoOverflow](#) = 5308
- [kStatus_FLEXCAN_RxFifoWarning](#) = 5309
- [kStatus_FLEXCAN_ErrorStatus](#) = 5310
- [kStatus_FLEXCAN_UnHandled](#) = 5311
- [kStatus_I2C_Busy](#) = 1100
- [kStatus_I2C_Idle](#) = 1101
- [kStatus_I2C_Nak](#) = 1102
- [kStatus_I2C_ArbitrationLost](#) = 1103
- [kStatus_I2C_Timeout](#) = 1104
- [kStatus_I2C_Addr_Nak](#) = 1105
- [kStatus_SAI_TxBusy](#) = 1900
- [kStatus_SAI_RxBusy](#) = 1901
- [kStatus_SAI_TxError](#) = 1902
- [kStatus_SAI_RxError](#) = 1903
- [kStatus_SAI_QueueFull](#) = 1904
- [kStatus_SAI_TxIdle](#) = 1905
- [kStatus_SAI_RxIdle](#) = 1906
- [kStatus_SMC_StopAbort](#) = 3900
- [kStatus_UART_TxBusy](#) = 1000
- [kStatus_UART_RxBusy](#) = 1001
- [kStatus_UART_TxIdle](#) = 1002

- `kStatus_UART_RxIdle` = 1003
- `kStatus_UART_TxWatermarkTooLarge` = 1004
- `kStatus_UART_RxWatermarkTooLarge` = 1005
- `kStatus_UART_FlagCannotClearManually` = 1006
- `kStatus_UART_Error` = 1007
- `kStatus_UART_RxRingBufferOverflow` = 1008
- `kStatus_UART_RxHardwareOverflow` = 1009
- `kStatus_UART_NoiseError` = 1010
- `kStatus_UART_FramingError` = 1011
- `kStatus_UART_ParityError` = 1012
- `kStatus_UART_BaudrateNotSupport` = 1013
- `kStatus_UART_IdleLineDetected` = 1014
- `kStatus_NOTIFIER_ErrorNotificationBefore` = 9800
- `kStatus_NOTIFIER_ErrorNotificationAfter` = 9801
- `kStatus_SDSPI_SetFrequencyFailed` = 2200
- `kStatus_SDSPI_ExchangeFailed` = 2201
- `kStatus_SDSPI_WaitReadyFailed` = 2202
- `kStatus_SDSPI_ResponseError` = 2203
- `kStatus_SDSPI_WriteProtected` = 2204
- `kStatus_SDSPI_GoIdleFailed` = 2205
- `kStatus_SDSPI_SendCommandFailed` = 2206
- `kStatus_SDSPI_ReadFailed` = 2207
- `kStatus_SDSPI_WriteFailed` = 2208
- `kStatus_SDSPI_SendInterfaceConditionFailed` = 2209
- `kStatus_SDSPI_SendOperationConditionFailed` = 2210
- `kStatus_SDSPI_ReadOcrFailed` = 2211
- `kStatus_SDSPI_SetBlockSizeFailed` = 2212
- `kStatus_SDSPI_SendCsdFailed` = 2213
- `kStatus_SDSPI_SendCidFailed` = 2214
- `kStatus_SDSPI_StopTransmissionFailed` = 2215
- `kStatus_SDSPI_SendApplicationCommandFailed` = 2216

Chapter 5

Deprecated List

Global **CS42888_SetFuncMode** (**cs42888_handle_t** *handle, **cs42888_func_mode** mode)
api, Do not use it anymore. It has been superceded by [CS42888_SelectFunctionalMode](#).

Global **flexcan_clock_source_t**

Do not use the kFLEXCAN_ClkSrcOs. It has been superceded kFLEXCAN_ClkSrc0

Do not use the kFLEXCAN_ClkSrcPeri. It has been superceded kFLEXCAN_ClkSrc1

Global **MMC_PowerOffCard** (**SDMMCHOST_TYPE** *base, const **sdkmmchost_pwr_card_t** *pwr)
Do not use this function. It has been superceded by [MMC_SetCardPower](#).

Global **MMC_PowerOnCard** (**SDMMCHOST_TYPE** *base, const **sdkmmchost_pwr_card_t** *pwr)
Do not use this function. It has been superceded by [MMC_SetCardPower](#).

Global **SAI_RxGetDefaultConfig** (**sai_config_t** *config)
Do not use this function. It has been superceded by [SAI_GetClassicI2SConfig](#), [SAI_GetLeftJustified-Config](#) , [SAI_GetRightJustifiedConfig](#), [SAI_GetDSPConfig](#), [SAI_GetTDMConfig](#)

Global **SAI_RxInit** (**I2S_Type** *base, const **sai_config_t** *config)
Do not use this function. It has been superceded by [SAI_Init](#)

Global **SAI_RxSetFormat** (**I2S_Type** *base, **sai_transfer_format_t** *format, **uint32_t** mclkSource-ClockHz, **uint32_t** bclkSourceClockHz)
Do not use this function. It has been superceded by [SAI_RxSetConfig](#)

Global **SAI_TransferRxSetFormat** (**I2S_Type** *base, **sai_handle_t** *handle, **sai_transfer_format_t** *format, **uint32_t** mclkSourceClockHz, **uint32_t** bclkSourceClockHz)
Do not use this function. It has been superceded by [SAI_TransferRxSetConfig](#)

Global **SAI_TransferTxSetFormat** (**I2S_Type** *base, **sai_handle_t** *handle, **sai_transfer_format_t** *format, **uint32_t** mclkSourceClockHz, **uint32_t** bclkSourceClockHz)
Do not use this function. It has been superceded by [SAI_TransferTxSetConfig](#)

Global **SAI_TxGetDefaultConfig** (**sai_config_t** *config)
Do not use this function. It has been superceded by [SAI_GetClassicI2SConfig](#), [SAI_GetLeftJustified-Config](#) , [SAI_GetRightJustifiedConfig](#), [SAI_GetDSPConfig](#), [SAI_GetTDMConfig](#)

Global **SAI_TxInit** (**I2S_Type** *base, const **sai_config_t** *config)
Do not use this function. It has been superceded by [SAI_Init](#)

Global **SAI_TxSetFormat** (**I2S_Type** *base, **sai_transfer_format_t** *format, **uint32_t** mclkSource-ClockHz, **uint32_t** bclkSourceClockHz)
Do not use this function. It has been superceded by [SAI_TxSetConfig](#)

Global **SD_Deinit** (**sd_card_t** *card)

Do not use this function. It has been superseded by [SD_HostDeinit](#), [SD_CardDeinit](#). This function deinitializes the specific card and host.

Global **SD_HostReset** (**SDMMCHOST_CONFIG** *host)

Do not use this function. It has been superseded by [SD_HostDoReset](#).

Global **SD_Init** (**sd_card_t** *card)

Do not use this function. It has been superseded by [SD_HostInit](#), [SD_CardInit](#).

Global **SD_PowerOffCard** (**SDMMCHOST_TYPE** *base, const **sdmmchost_pwr_card_t** *pwr)

Do not use this function. It has been superseded by [SD_SetCardPower](#).

Global **SD_PowerOnCard** (**SDMMCHOST_TYPE** *base, const **sdmmchost_pwr_card_t** *pwr)

Do not use this function. It has been superseded by [SD_SetCardPower](#).

Global **SD_WaitCardDetectStatus** (**SDMMCHOST_TYPE** *hostBase, const **sdmmchost_detect_card_t** *cd, bool waitCardStatus)

Do not use this function. It has been superseded by [SD_PollingCardInsert](#).

Global **SDIO_HostReset** (**SDMMCHOST_CONFIG** *host)

Do not use this function. It has been superseded by [SDIO_HostDoReset](#).

Global **SDIO_PowerOffCard** (**SDMMCHOST_TYPE** *base, const **sdmmchost_pwr_card_t** *pwr)

Do not use this function. It has been superseded by [SDIO_SetCardPower](#).

Global **SDIO_PowerOnCard** (**SDMMCHOST_TYPE** *base, const **sdmmchost_pwr_card_t** *pwr)

Do not use this function. It has been superseded by [SDIO_SetCardPower](#).

Global **SDIO_WaitCardDetectStatus** (**SDMMCHOST_TYPE** *hostBase, const **sdmmchost_detect_card_t** *cd, bool waitCardStatus)

Do not use this function. It has been superseded by [SDIO_PollingCardInsert](#).

Global **SDMMCHOST_PowerOffCard** (**SDMMCHOST_TYPE** *base, const **sdmmchost_pwr_card_t** *pwr)

Do not use this function. It has been superseded by [SDMMCHOST_SetCardPower..](#)

Global **SDMMCHOST_PowerOnCard** (**SDMMCHOST_TYPE** *base, const **sdmmchost_pwr_card_t** *pwr)

Do not use this function. It has been superseded by [SDMMCHOST_SetCardPower..](#)

Class **sdmmchost_pwr_card_t**

Do not use this structure anymore.

Global **SDMMCHOST_Reset** (**sdmmchost_t** *host)

Do not use this function. Application should not call this function, driver is responsible for the host reset..

Global [SDMMCHOST_WaitCardDetectStatus](#) (SDMMCHOST_TYPE *hostBase, const sdmmchost_detect_card_t *cd, bool waitCardStatus)

Do not use this function. It has been superseded by [SDMMCHOST_PollingCardDetectStatus..](#)

Global [WM8904_SetMasterSlave](#) (wm8904_handle_t *handle, bool master)

DO NOT USE THIS API ANYMORE. IT HAS BEEN SUPERCEDED BY [WM8904_SeMasterClock](#)



Chapter 6 Clock Driver

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 ENET_CLOCKS`

Overview

- *Clock ip name array for ENET.*
• #define [PORT_CLOCKS](#)
- *Clock ip name array for PORT.*
• #define [SAI_CLOCKS](#)
- *Clock ip name array for SAI.*
• #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 [SDHC_CLOCKS](#)
- *Clock ip name array for SDHC.*
• #define [FTM_CLOCKS](#)
- *Clock ip name array for FTM.*
• #define [EDMA_CLOCKS](#)
- *Clock ip name array for EDMA.*
• #define [FLEXCAN_CLOCKS](#)
- *Clock ip name array for FLEXCAN.*
• #define [DAC_CLOCKS](#)
- *Clock ip name array for DAC.*
• #define [ADC16_CLOCKS](#)
- *Clock ip name array for ADC16.*
• #define [SYSPMU_CLOCKS](#)
- *Clock ip name array for MPU.*
• #define [VREF_CLOCKS](#)
- *Clock ip name array for VREF.*
• #define [CMT_CLOCKS](#)
- *Clock ip name array for CMT.*
• #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 [PDB_CLOCKS](#)
- *Clock ip name array for PDB.*
• #define [FTF_CLOCKS](#)
- *Clock ip name array for FTF.*
• #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_EREFS0_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_EREFS0_MASK` }
OSCERCLK enable mode.
- enum `mcg_fll_src_t` {

Overview

- `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` }
 - MCG PLLCS select.*
- 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_Irc1kEnable = MCG_C1_IRCLKEN_MASK,
kMCG_Irc1kEnableInStop = 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_SetEr32kClock` (uint32_t src)
Set ERCLK32K source.
- static void `CLOCK_SetSdhc0Clock` (uint32_t src)
Set SDHC0 clock source.
- static void `CLOCK_SetEnetTime0Clock` (uint32_t src)
Set enet timestamp clock source.
- static void `CLOCK_SetRmii0Clock` (uint32_t src)

Overview

- *Set RMII clock source.*
- static void [CLOCK_SetTraceClock](#) (uint32_t src)
Set debug trace clock source.
- static void [CLOCK_SetPllFllSelClock](#) (uint32_t src)
Set PLLFLLSEL clock source.
- 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[PLLFLLSEL].
- uint32_t [CLOCK_GetEr32kClkFreq](#) (void)
Get the external reference 32K clock frequency (ERCLK32K).
- uint32_t [CLOCK_GetOsc0ErClkFreq](#) (void)
Get the OSC0 external reference clock frequency (OSC0ERCLK).
- 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, 1))
CLOCK driver version 2.5.1.

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 fcrdiv)
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, oscr_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)

Overview

- *Initializes the OSC0.*
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 boot up.
- [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.

Data Structure Documentation

6.2.1 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.

6.2.1.0.0.1 Field Documentation

6.2.1.0.0.1.1 [uint8_t sim_clock_config_t::pllFllSel](#)

6.2.1.0.0.1.2 [uint8_t sim_clock_config_t::er32kSrc](#)

6.2.1.0.0.1.3 [uint32_t sim_clock_config_t::clkdiv1](#)

6.2.2 struct oscer_config_t

Data Fields

- [uint8_t enableMode](#)
OSCEERCLK enable mode.

6.2.2.0.0.2 Field Documentation

6.2.2.0.0.2.1 [uint8_t oscer_config_t::enableMode](#)

OR'ed value of [_oscer_enable_mode](#).

Data Structure Documentation

6.2.3 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.

6.2.3.0.0.3 Field Documentation

6.2.3.0.0.3.1 uint32_t osc_config_t::freq

6.2.3.0.0.3.2 uint8_t osc_config_t::capLoad

6.2.3.0.0.3.3 osc_mode_t osc_config_t::workMode

6.2.3.0.0.3.4 oscer_config_t osc_config_t::oscerConfig

6.2.4 struct mcg_pll_config_t

Data Fields

- uint8_t [enableMode](#)
Enable mode.
- uint8_t [prdiv](#)
Reference divider PRDIV.
- uint8_t [vdiv](#)
VCO divider VDIV.

6.2.4.0.0.4 Field Documentation

6.2.4.0.0.4.1 uint8_t mcg_pll_config_t::enableMode

OR'ed value of enumeration `_mcg_pll_enable_mode`.

6.2.4.0.0.4.2 `uint8_t mcg_pll_config_t::prdiv`

6.2.4.0.0.4.3 `uint8_t mcg_pll_config_t::vdiv`

6.2.5 `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.

Macro Definition Documentation

6.2.5.0.0.5 Field Documentation

6.2.5.0.0.5.1 `mcg_mode_t mcg_config_t::mcgMode`

6.2.5.0.0.5.2 `uint8_t mcg_config_t::irclkEnableMode`

6.2.5.0.0.5.3 `mcg_irc_mode_t mcg_config_t::ircs`

6.2.5.0.0.5.4 `uint8_t mcg_config_t::fcrdiv`

6.2.5.0.0.5.5 `uint8_t mcg_config_t::frdiv`

6.2.5.0.0.5.6 `mcg_drs_t mcg_config_t::drs`

6.2.5.0.0.5.7 `mcg_dmx32_t mcg_config_t::dmx32`

6.2.5.0.0.5.8 `mcg_oscsel_t mcg_config_t::oscsel`

6.2.5.0.0.5.9 `mcg_pll_config_t mcg_config_t::pll0Config`

Macro Definition Documentation

6.3.1 `#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[OSCSSEL] 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.

6.3.2 `#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.

6.3.3 #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))**6.3.4 #define MCG_INTERNAL_IRC_48M 48000000U****6.3.5 #define DMAMUX_CLOCKS****Value:**

```
{
    \
    kCLOCK_Dmamux0 \
}
```

6.3.6 #define RTC_CLOCKS**Value:**

```
{
    \
    kCLOCK_Rtc0 \
}
```

6.3.7 #define ENET_CLOCKS**Value:**

```
{
    \
    kCLOCK_Enet0 \
}
```

6.3.8 #define PORT_CLOCKS**Value:**

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

6.3.9 #define SAI_CLOCKS**Value:**

```
{
    \
    kCLOCK_Sai0 \
}
```

6.3.10 #define FLEXBUS_CLOCKS

Value:

```
{  
    \kCLOCK_Flexbus0 \  
}
```

6.3.11 #define EWM_CLOCKS

Value:

```
{  
    \kCLOCK_Ewm0 \  
}
```

6.3.12 #define PIT_CLOCKS

Value:

```
{  
    \kCLOCK_Pit0 \  
}
```

6.3.13 #define DSPI_CLOCKS

Value:

```
{  
    \kCLOCK_Spi0, kCLOCK_Spi1, kCLOCK_Spi2 \  
}
```

6.3.14 #define LPTMR_CLOCKS

Value:

```
{  
    \kCLOCK_Lptmr0 \  
}
```

6.3.15 #define SDHC_CLOCKS

Value:

```
{  
    \kCLOCK_Sdhc0 \  
}
```

6.3.16 #define FTM_CLOCKS

Value:

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

6.3.17 #define EDMA_CLOCKS

Value:

```
{  
    \kCLOCK_Dma0 \  
}
```

6.3.18 #define FLEXCAN_CLOCKS

Value:

```
{  
    \kCLOCK_Flexcan0 \  
}
```

6.3.19 #define DAC_CLOCKS

Value:

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

6.3.20 #define ADC16_CLOCKS

Value:

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

6.3.21 #define SYSMPU_CLOCKS

Value:

```
{  
    \kCLOCK_Sysmpu0 \  
}
```

6.3.22 #define VREF_CLOCKS

Value:

```
{  
    \kCLOCK_Vref0 \  
}
```

6.3.23 #define CMT_CLOCKS

Value:

```
{  
    \kCLOCK_Cmt0 \  
}
```

6.3.24 #define UART_CLOCKS

Value:

```
{  
    \kCLOCK_Uart0, kCLOCK_Uart1, kCLOCK_Uart2, kCLOCK_Uart3, kCLOCK_Uart4, kCLOCK_Uart5 \  
}
```


6.3.25 #define RNGA_CLOCKS

Value:

```
{  
    \kCLOCK_Rnga0 \  
}
```

6.3.26 #define CRC_CLOCKS

Value:

```
{  
    \kCLOCK_Crc0 \  
}
```

6.3.27 #define I2C_CLOCKS

Value:

```
{  
    \kCLOCK_I2c0, kCLOCK_I2c1, kCLOCK_I2c2 \  
}
```

6.3.28 #define PDB_CLOCKS

Value:

```
{  
    \kCLOCK_Pdb0 \  
}
```

6.3.29 #define FTF_CLOCKS

Value:

```
{  
    \kCLOCK_Ftf0 \  
}
```

Enumeration Type Documentation

6.3.30 #define CMP_CLOCKS

Value:

```
{  
    kCLOCK_Cmp0, kCLOCK_Cmp1, kCLOCK_Cmp2 \
```

6.3.31 #define SYS_CLK kCLOCK_CoreSysClk

Enumeration Type Documentation

6.4.1 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[PLL/FLLSEL].
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.

6.4.2 enum clock_usb_src_t

Enumerator

kCLOCK_UsbSrcPll0 Use PLL0.
kCLOCK_UsbSrcIrc48M Use IRC48M.
kCLOCK_UsbSrcExt Use USB_CLKIN.

6.4.3 enum clock_ip_name_t

6.4.4 enum osc_mode_t

Enumerator

- kOSC_ModeExt* Use an external clock.
- kOSC_ModeOscLowPower* Oscillator low power.
- kOSC_ModeOscHighGain* Oscillator high gain.

6.4.5 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

6.4.6 enum _oscer_enable_mode

Enumerator

- kOSC_ErClkEnable* Enable.
- kOSC_ErClkEnableInStop* Enable in stop mode.

6.4.7 enum mcg_fl_src_t

Enumerator

- kMCG_FlSrcExternal* External reference clock is selected.
- kMCG_FlSrcInternal* The slow internal reference clock is selected.

6.4.8 enum mcg_irc_mode_t

Enumerator

- kMCG_IrcSlow* Slow internal reference clock selected.
- kMCG_IrcFast* Fast internal reference clock selected.

Enumeration Type Documentation

6.4.9 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.

6.4.10 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.

6.4.11 enum mcg_pll_ref_src_t

Enumerator

kMCG_PllRefOsc0 Selects OSC0 as PLL reference clock.

kMCG_PllRefOsc1 Selects OSC1 as PLL reference clock.

6.4.12 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.

6.4.13 enum mcg_atm_select_t

Enumerator

kMCG_AtmSel32k 32 kHz Internal Reference Clock selected

kMCG_AtmSel4m 4 MHz Internal Reference Clock selected

6.4.14 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.

6.4.15 enum mcg_pll_clk_select_t

Enumerator

kMCG_PllClkSelPll0 PLL0 output clock is selected.

6.4.16 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.

6.4.17 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.

6.4.18 anonymous enum

Enumeration _mcg_status_flags_t

Enumerator

kMCG_Osc0LostFlag OSC0 lost.

Function Documentation

kMCG_Osc0InitFlag OSC0 crystal initialized.

kMCG_RtcOscLostFlag RTC OSC lost.

kMCG_Pll0LostFlag PLL0 lost.

kMCG_Pll0LockFlag PLL0 locked.

6.4.19 anonymous enum

Enumeration `_mcg_ircclk_enable_mode`

Enumerator

kMCG_IrcclkEnable MCGIRCLK enable.

kMCG_IrcclkEnableInStop MCGIRCLK enable in stop mode.

6.4.20 anonymous enum

Enumeration `_mcg_pll_enable_mode`

Enumerator

kMCG_PllEnableIndependent 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_PllEnableInStop MCGPLLCLK enable in STOP mode.

6.4.21 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.

Function Documentation

**6.5.1 static void CLOCK_EnableClock (clock_ip_name_t name) [inline],
[static]**

Parameters

<i>name</i>	Which clock to enable, see clock_ip_name_t .
-------------	--

6.5.2 static void CLOCK_DisableClock (clock_ip_name_t *name*) [inline], [static]

Parameters

<i>name</i>	Which clock to disable, see clock_ip_name_t .
-------------	---

6.5.3 static void CLOCK_SetEr32kClock (uint32_t *src*) [inline], [static]

Parameters

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

6.5.4 static void CLOCK_SetSdhc0Clock (uint32_t *src*) [inline], [static]

Parameters

<i>src</i>	The value to set SDHC0 clock source.
------------	--------------------------------------

6.5.5 static void CLOCK_SetEnetTime0Clock (uint32_t *src*) [inline], [static]

Parameters

<i>src</i>	The value to set enet timestamp clock source.
------------	---

6.5.6 static void CLOCK_SetRmii0Clock (uint32_t *src*) [inline], [static]

Function Documentation

Parameters

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

6.5.7 static void CLOCK_SetTraceClock (uint32_t *src*) [inline], [static]

Parameters

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

6.5.8 static void CLOCK_SetPIIFllSelClock (uint32_t *src*) [inline], [static]

Parameters

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

6.5.9 static void CLOCK_SetClkOutClock (uint32_t *src*) [inline], [static]

Parameters

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

6.5.10 static void CLOCK_SetRtcClkOutClock (uint32_t *src*) [inline], [static]

Parameters

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

6.5.11 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 <i>src</i> .

Return values

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

6.5.12 static void CLOCK_DisableUsbfs0Clock (void) [inline], [static]

Disable USB FS clock.

6.5.13 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.

6.5.14 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

Function Documentation

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

Returns

Clock frequency value in Hertz

6.5.15 uint32_t CLOCK_GetCoreSysClkFreq (void)

Returns

Clock frequency in Hz.

6.5.16 uint32_t CLOCK_GetPlatClkFreq (void)

Returns

Clock frequency in Hz.

6.5.17 uint32_t CLOCK_GetBusClkFreq (void)

Returns

Clock frequency in Hz.

6.5.18 uint32_t CLOCK_GetFlexBusClkFreq (void)

Returns

Clock frequency in Hz.

6.5.19 uint32_t CLOCK_GetFlashClkFreq (void)

Returns

Clock frequency in Hz.

6.5.20 uint32_t CLOCK_GetPIIFllSelClkFreq (void)

Returns

Clock frequency in Hz.

6.5.21 uint32_t CLOCK_GetEr32kClkFreq (void)

Returns

Clock frequency in Hz.

6.5.22 uint32_t CLOCK_GetOsc0ErClkFreq (void)

Returns

Clock frequency in Hz.

6.5.23 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.
---------------	-------------------------------------

6.5.24 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.

6.5.25 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.

6.5.26 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.

6.5.27 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.

6.5.28 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 MCGFFCLK.

6.5.29 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.

6.5.30 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.
---------------	---

6.5.31 **status_t CLOCK_SetInternalRefClkConfig (uint8_t enableMode, mcg_irc_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 <code>_mcg_ircclk_enable_mode</code> .
<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.

6.5.32 **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

Function Documentation

<i>kStatus_MCG_Source-Used</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.

6.5.33 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].
--------------	--

6.5.34 void CLOCK_EnablePll0 (mcg_pll_config_t const * *config*)

This function sets up 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.
---------------	---

6.5.35 static void CLOCK_DisablePll0 (void) [inline], [static]

This function disables the PLL0 in FLL mode. It should be used together with the [CLOCK_EnablePll0](#).

6.5.36 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.

6.5.37 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.
-------------	----------------------

6.5.38 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.
-------------	----------------------

6.5.39 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.
-------------	----------------------

6.5.40 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:

Function Documentation

```
* 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`.

6.5.41 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

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

6.5.42 static void OSC_SetExtRefClkConfig (OSC_Type * *base*, oscr_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.

6.5.43 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 _osc_cap_load .

Example:

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

6.5.44 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.
---------------	--

6.5.45 void CLOCK_DeinitOsc0 (void)

This function deinitializes the OSC0.

6.5.46 static void CLOCK_SetXtal0Freq (uint32_t *freq*) [inline], [static]

Function Documentation

Parameters

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

6.5.47 static void CLOCK_SetXtal32Freq (uint32_t *freq*) [inline], [static]

Parameters

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

6.5.48 void CLOCK_SetSlowIrcFreq (uint32_t *freq*)

Parameters

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

6.5.49 void CLOCK_SetFastIrcFreq (uint32_t *freq*)

Parameters

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

6.5.50 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.

6.5.51 `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](#).

6.5.52 `status_t` **CLOCK_SetFeiMode (mcg_dmx32_t *dmx32*, mcg_drs_t *drs*, void(*)*(void) fillStableDelay*)**

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.

Function Documentation

<i>flStableDelay</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.

6.5.53 `status_t CLOCK_SetFeeMode (uint8_t frdiv, mcg_dmx32_t dmx32, mcg_drs_t drs, void(*)(void) flStableDelay)`

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>flStableDelay</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.

6.5.54 `status_t CLOCK_SetFbiMode (mcg_dmx32_t dmx32, mcg_drs_t drs, void(*)(void) flStableDelay)`

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.

6.5.55 **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.
-------------------------------------	--------------------------------------

Function Documentation

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

6.5.56 **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.

6.5.57 **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.

6.5.58 **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 `pllcs` 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_PllClkSelExtPll, NULL);`

6.5.59 `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.

6.5.60 `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(...);
*
```

Function Documentation

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.

6.5.61 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.

6.5.62 status_t CLOCK_BootToFeiMode (mcg_dm32_t dm32, mcg_drs_t drs, void(*) (void) flStableDelay)

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>dm32</i>	DMX32 in FEI mode.
<i>drs</i>	The DCO range selection.
<i>flStableDelay</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.

6.5.63 **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.

6.5.64 **status_t CLOCK_BootToBlpiMode (uint8_t fcrdiv, 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.

Function Documentation

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 <code>_mcg_ircclk_enable_mode</code> .

Return values

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

6.5.65 `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_Mode-Unreachable</i>	Could not switch to the target mode.
<i>kStatus_Success</i>	Switched to the target mode successfully.

6.5.66 `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.

6.5.67 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.

Variable Documentation

6.6.1 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(8000000);
*
```

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.

6.6.2 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.

Multipurpose Clock Generator (MCG)

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

6.7.1 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.

6.7.1.1 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.

6.7.1.2 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 divider. 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.

Multipurpose Clock Generator (MCG)

6.7.1.3 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.

6.7.1.4 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.

6.7.1.5 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.

6.7.1.6 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.

6.7.2 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.

6.7.2.1 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>flStableDelay=NULL</code>
	FBI -> BLPI	<code>CLOCK_SetLowPowerEnable(true)</code>

6.7.2.2 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>flStableDelay=NULL</code>
	FBI -> BLPI	<code>CLOCK_SetLowPowerEnable(true)</code>

Multipurpose Clock Generator (MCG)

6.7.2.3 Switch between BLPI and PEE

Use case	Steps	Functions
BLPI -> PEE	BLPI -> FBI	CLOCK_InternalModeToFbiModeQuick(...)
	Change external clock source if need	CLOCK_SetExternalRefClkConfig(...)
	FBI -> FBE	CLOCK_SetFbeMode(...) // flStableDelay=NULL
	FBE -> PBE	CLOCK_SetPbeMode(...)
	PBE -> PEE	CLOCK_SetPeeMode(...)
PEE -> BLPI	PEE -> FBE	CLOCK_ExternalModeToFbeModeQuick(...)
	Configure MCGIRCLK if need	CLOCK_SetInternalRefClkConfig(...)
	FBE -> FBI	CLOCK_SetFbiMode(...) with flStableDelay=NULL
	FBI -> BLPI	CLOCK_SetLowPowerEnable(true)

6.7.2.4 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_ExternalModeToFbeModeQuick(...)
	FBE -> BLPE	CLOCK_SetLowPowerEnable(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_ExternalModeToFbeModeQuick(...)

Multipurpose Clock Generator (MCG)

	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)

6.7.2.5 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(...)

Multipurpose Clock Generator (MCG)

	FBE -> FBI	CLOCK_SetFbiMode(...) with fllStableDelay=NULL
	Change source	CLOCK_SetExternalRefClk-Config(...)
	FBI -> FEE	CLOCK_SetFeeMode(...)
FEE -> BLPE	FEE -> FBI	CLOCK_SetFbiMode(...) with fllStableDelay=NULL
	Change source	CLOCK_SetExternalRefClk-Config(...)
	PBI -> FBE	CLOCK_SetFbeMode(...) with fllStableDelay=NULL
	FBE -> BLPE	CLOCK_SetLowPower-Enable(true)

6.7.2.6 Switch between BLPI and PEI

Use case	Steps	Functions
BLPI -> PEI	BLPI -> PBI	CLOCK_SetPbiMode(...)
	PBI -> PEI	CLOCK_SetPeiMode(...)
	Configure MCGIRCLK if need	CLOCK_SetInternalRefClk-Config(...)
PEI -> BLPI	Configure MCGIRCLK if need	CLOCK_SetInternalRefClk-Config
	PEI -> FBI	CLOCK_InternalModeToFbi-ModeQuick(...)
	FBI -> BLPI	CLOCK_SetLowPower-Enable(true)

6.7.3 Code Configuration Option

6.7.3.1 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 his 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 7

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

Overview

The MCUXpresso SDK provides a peripheral driver for the 16-bit SAR Analog-to-Digital Converter (ADC16) module of MCUXpresso SDK devices.

Typical use case

7.2.1 Polling Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/adc16

7.2.2 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_COCO_MASK }
 - enum [_adc16_status_flags](#) {
[kADC16_ActiveFlag](#) = ADC_SC2_ADACT_MASK,
[kADC16_CalibrationFailedFlag](#) = ADC_SC3_CALF_MASK }
 - enum [adc16_channel_mux_mode_t](#) {
[kADC16_ChannelMuxA](#) = 0U,
[kADC16_ChannelMuxB](#) = 1U }
- Channel multiplexer mode for each channel.*

Typical use case

- 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, 1, 0)`)
ADC16 driver version 2.1.0.

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.

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)
Gets the conversion value.
- `uint32_t` `ADC16_GetChannelStatusFlags` (`ADC_Type *base`, `uint32_t` channelGroup)

Data Structure Documentation

Gets the status flags of channel.

Data Structure Documentation

7.3.1 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.

7.3.1.0.0.6 Field Documentation**7.3.1.0.0.6.1** `adc16_reference_voltage_source_t` `adc16_config_t::referenceVoltageSource`**7.3.1.0.0.6.2** `adc16_clock_source_t` `adc16_config_t::clockSource`**7.3.1.0.0.6.3** `bool` `adc16_config_t::enableAsynchronousClock`**7.3.1.0.0.6.4** `adc16_clock_divider_t` `adc16_config_t::clockDivider`**7.3.1.0.0.6.5** `adc16_resolution_t` `adc16_config_t::resolution`**7.3.1.0.0.6.6** `adc16_long_sample_mode_t` `adc16_config_t::longSampleMode`**7.3.1.0.0.6.7** `bool` `adc16_config_t::enableHighSpeed`**7.3.1.0.0.6.8** `bool` `adc16_config_t::enableLowPower`**7.3.1.0.0.6.9** `bool` `adc16_config_t::enableContinuousConversion`**7.3.2 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.

7.3.2.0.0.7 Field Documentation**7.3.2.0.0.7.1** `adc16_hardware_compare_mode_t` `adc16_hardware_compare_config_t::hardwareCompareMode`See "`adc16_hardware_compare_mode_t`".**7.3.2.0.0.7.2** `int16_t` `adc16_hardware_compare_config_t::value1`**7.3.2.0.0.7.3** `int16_t` `adc16_hardware_compare_config_t::value2`**7.3.3 struct `adc16_channel_config_t`****Data Fields**

- `uint32_t` `channelNumber`
Setting the conversion channel number.
- `bool` `enableInterruptOnConversionCompleted`

Enumeration Type Documentation

- *Generate an interrupt request once the conversion is completed.*
• bool [enableDifferentialConversion](#)
Using Differential sample mode.

7.3.3.0.0.8 Field Documentation

7.3.3.0.0.8.1 uint32_t adc16_channel_config_t::channelNumber

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

7.3.3.0.0.8.2 bool adc16_channel_config_t::enableInterruptOnConversionCompleted

7.3.3.0.0.8.3 bool adc16_channel_config_t::enableDifferentialConversion

Macro Definition Documentation

7.4.1 #define FSL_ADC16_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))

Enumeration Type Documentation

7.5.1 enum _adc16_channel_status_flags

Enumerator

kADC16_ChannelConversionDoneFlag Conversion done.

7.5.2 enum _adc16_status_flags

Enumerator

kADC16_ActiveFlag Converter is active.

kADC16_CalibrationFailedFlag Calibration is failed.

7.5.3 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.

7.5.4 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.

7.5.5 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.

7.5.6 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.

7.5.7 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.

Function Documentation

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.

7.5.8 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.

7.5.9 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.

7.5.10 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 $x < \text{value1} \parallel x > \text{value2}$; else, $\text{value1} > x > \text{value2}$.

kADC16_HardwareCompareMode3 if $\text{value1} \leq \text{value2}$, then $\text{value1} \leq x \leq \text{value2}$; else $x > \text{value1} \parallel x \leq \text{value2}$.

Function Documentation

7.6.1 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".

7.6.2 void ADC16_Deinit (ADC_Type * *base*)

Parameters

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

7.6.3 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   = true;
*  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.
---------------	---

7.6.4 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.

Function Documentation

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.

7.6.5 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.

7.6.6 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.

7.6.7 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.

7.6.8 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".

7.6.9 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.

7.6.10 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.

Function Documentation

Parameters

<i>base</i>	ADC16 peripheral base address.
<i>mode</i>	Setting the hardware average mode. See "adc16_hardware_average_mode_t".

7.6.11 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".

7.6.12 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".

7.6.13 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.

Function Documentation

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.

7.6.14 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.

7.6.15 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 8

CMP: Analog Comparator Driver

Overview

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

Typical use case

8.2.1 Polling Configuration

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

8.2.2 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.

Typical use case

- 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.
- static void `CMP_EnablePassThroughMode` (`CMP_Type *base`, bool enable)
 Enables/disables the pass through mode.
- void `CMP_SetFilterConfig` (`CMP_Type *base`, const `cmp_filter_config_t *config`)
 Configures the filter.
- void `CMP_SetDACConfig` (`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.

Data Structure Documentation

8.3.1 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.

8.3.1.0.0.9 Field Documentation

8.3.1.0.0.9.1 bool cmp_config_t::enableCmp

8.3.1.0.0.9.2 cmp_hysteresis_mode_t cmp_config_t::hysteresisMode

8.3.1.0.0.9.3 bool cmp_config_t::enableHighSpeed

8.3.1.0.0.9.4 bool cmp_config_t::enableInvertOutput

8.3.1.0.0.9.5 bool cmp_config_t::useUnfilteredOutput

8.3.1.0.0.9.6 bool cmp_config_t::enablePinOut

8.3.2 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.

Enumeration Type Documentation

8.3.2.0.0.10 Field Documentation

8.3.2.0.0.10.1 `bool cmp_filter_config_t::enableSample`

8.3.2.0.0.10.2 `uint8_t cmp_filter_config_t::filterCount`

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

8.3.2.0.0.10.3 `uint8_t cmp_filter_config_t::filterPeriod`

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

8.3.3 `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.

8.3.3.0.0.11 Field Documentation

8.3.3.0.0.11.1 `cmp_reference_voltage_source_t cmp_dac_config_t::referenceVoltageSource`

8.3.3.0.0.11.2 `uint8_t cmp_dac_config_t::DACValue`

Available range is 0-63.

Macro Definition Documentation

8.4.1 `#define FSL_CMP_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

Enumeration Type Documentation

8.5.1 `enum _cmp_interrupt_enable`

Enumerator

- `kCMP_OutputRisingInterruptEnable`* Comparator interrupt enable rising.
- `kCMP_OutputFallingInterruptEnable`* Comparator interrupt enable falling.

8.5.2 `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.

8.5.3 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.

8.5.4 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.

Function Documentation

8.6.1 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.

8.6.2 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.

Function Documentation

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.
-------------	------------------------------

8.6.3 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.

8.6.4 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.
---------------	---

8.6.5 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.

8.6.6 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.

8.6.7 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.

8.6.8 static void CMP_EnablePassThroughMode (CMP_Type * *base*, bool *enable*) [inline], [static]

Parameters

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

Function Documentation

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

8.6.9 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.

8.6.10 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.

8.6.11 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".

8.6.12 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".
-------------	---

8.6.13 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".

8.6.14 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 9

CMT: Carrier Modulator Transmitter Driver

Overview

The carrier modulator transmitter (CMT) module provides the means to generate the protocol timing and carrier signals for a side variety of encoding schemes. The CMT incorporates hardware to off-load the critical and/or lengthy timing requirements associated with signal generation from the CPU. The MCU-Xpresso SDK provides a driver for the CMT module of the MCUXpresso SDK devices.

Clock formulas

The CMT module has internal clock dividers. It was originally designed to be based on an 8 MHz bus clock that can be divided by 1, 2, 4, or 8 according to the specification. To be compatible with a higher bus frequency, the primary prescaler (PPS) was developed to receive a higher frequency and generate a clock enable signal called an intermediate frequency (IF). The IF must be approximately equal to 8 MHz and works as a clock enable to the secondary prescaler. For the PPS, the prescaler is selected according to the bus clock to generate an intermediate clock approximate to 8 MHz and is selected as $(\text{bus_clock_hz}/8000000)$. The secondary prescaler is the "cmtDivider". The clocks for the CMT module are listed below.

1. CMT clock frequency = $\text{bus_clock_Hz} / (\text{bus_clock_Hz} / 8000000) / \text{cmtDivider}$
2. CMT carrier and generator frequency = $\text{CMT clock frequency} / (\text{highCount1} + \text{lowCount1})$
(In FSK mode, the second frequency = $\text{CMT clock frequency} / (\text{highCount2} + \text{lowCount2})$)
3. CMT infrared output signal frequency
 - a. In Time and Baseband mode
CMT IRO signal mark time = $(\text{markCount} + 1) / (\text{CMT clock frequency} / 8)$
CMT IRO signal space time = $\text{spaceCount} / (\text{CMT clock frequency} / 8)$
 - b. In FSK mode
CMT IRO signal mark time = $(\text{markCount} + 1) / \text{CMT carrier and generator frequency}$
CMT IRO signal space time = $\text{spaceCount} / \text{CMT carrier and generator frequency}$

Typical use case

This is an example code to initialize data.

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/cmt`
This is an example IRQ handler to change the mark and space count to complete data modulation.

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

Data Structures

- struct `cmt_modulate_config_t`
CMT carrier generator and modulator configuration structure. [More...](#)

Typical use case

- struct `cmt_config_t`
CMT basic configuration structure. [More...](#)

Enumerations

- enum `cmt_mode_t` {
 `kCMT_DirectIROCtl` = 0x00U,
 `kCMT_TimeMode` = 0x01U,
 `kCMT_FSKMode` = 0x05U,
 `kCMT_BasebandMode` = 0x09U }
The modes of CMT.
 - enum `cmt_primary_clkdiv_t` {
 `kCMT_PrimaryClkDiv1` = 0U,
 `kCMT_PrimaryClkDiv2` = 1U,
 `kCMT_PrimaryClkDiv3` = 2U,
 `kCMT_PrimaryClkDiv4` = 3U,
 `kCMT_PrimaryClkDiv5` = 4U,
 `kCMT_PrimaryClkDiv6` = 5U,
 `kCMT_PrimaryClkDiv7` = 6U,
 `kCMT_PrimaryClkDiv8` = 7U,
 `kCMT_PrimaryClkDiv9` = 8U,
 `kCMT_PrimaryClkDiv10` = 9U,
 `kCMT_PrimaryClkDiv11` = 10U,
 `kCMT_PrimaryClkDiv12` = 11U,
 `kCMT_PrimaryClkDiv13` = 12U,
 `kCMT_PrimaryClkDiv14` = 13U,
 `kCMT_PrimaryClkDiv15` = 14U,
 `kCMT_PrimaryClkDiv16` = 15U }
The CMT clock divide primary prescaler.
 - enum `cmt_second_clkdiv_t` {
 `kCMT_SecondClkDiv1` = 0U,
 `kCMT_SecondClkDiv2` = 1U,
 `kCMT_SecondClkDiv4` = 2U,
 `kCMT_SecondClkDiv8` = 3U }
The CMT clock divide secondary prescaler.
 - enum `cmt_infrared_output_polarity_t` {
 `kCMT_IROActiveLow` = 0U,
 `kCMT_IROActiveHigh` = 1U }
The CMT infrared output polarity.
 - enum `cmt_infrared_output_state_t` {
 `kCMT_IROCtlLow` = 0U,
 `kCMT_IROCtlHigh` = 1U }
The CMT infrared output signal state control.
 - enum `_cmt_interrupt_enable` { `kCMT_EndOfCycleInterruptEnable` = CMT_MSC_EOCIE_MASK
}
- CMT interrupt configuration structure, default settings all disabled.*

Driver version

- #define `FSL_CMT_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 3)`)
CMT driver version 2.0.3.

Initialization and deinitialization

- void `CMT_GetDefaultConfig` (`cmt_config_t` *config)
Gets the CMT default configuration structure.
- void `CMT_Init` (`CMT_Type` *base, const `cmt_config_t` *config, `uint32_t` busClock_Hz)
Initializes the CMT module.
- void `CMT_Deinit` (`CMT_Type` *base)
Disables the CMT module and gate control.

Basic Control Operations

- void `CMT_SetMode` (`CMT_Type` *base, `cmt_mode_t` mode, `cmt_modulate_config_t` *modulate-Config)
Selects the mode for CMT.
- `cmt_mode_t` `CMT_GetMode` (`CMT_Type` *base)
Gets the mode of the CMT module.
- `uint32_t` `CMT_GetCMTFrequency` (`CMT_Type` *base, `uint32_t` busClock_Hz)
Gets the actual CMT clock frequency.
- static void `CMT_SetCarrirGenerateCountOne` (`CMT_Type` *base, `uint32_t` highCount, `uint32_t` lowCount)
Sets the primary data set for the CMT carrier generator counter.
- static void `CMT_SetCarrirGenerateCountTwo` (`CMT_Type` *base, `uint32_t` highCount, `uint32_t` lowCount)
Sets the secondary data set for the CMT carrier generator counter.
- void `CMT_SetModulateMarkSpace` (`CMT_Type` *base, `uint32_t` markCount, `uint32_t` spaceCount)
Sets the modulation mark and space time period for the CMT modulator.
- static void `CMT_EnableExtendedSpace` (`CMT_Type` *base, bool enable)
Enables or disables the extended space operation.
- void `CMT_SetIroState` (`CMT_Type` *base, `cmt_infrared_output_state_t` state)
Sets the IRO (infrared output) signal state.
- static void `CMT_EnableInterrupts` (`CMT_Type` *base, `uint32_t` mask)
Enables the CMT interrupt.
- static void `CMT_DisableInterrupts` (`CMT_Type` *base, `uint32_t` mask)
Disables the CMT interrupt.
- static `uint32_t` `CMT_GetStatusFlags` (`CMT_Type` *base)
Gets the end of the cycle status flag.

Data Structure Documentation

9.4.1 struct cmt_modulate_config_t

Data Fields

- `uint8_t` `highCount1`
The high-time for carrier generator first register.

Data Structure Documentation

- `uint8_t lowCount1`
The low-time for carrier generator first register.
- `uint8_t highCount2`
The high-time for carrier generator second register for FSK mode.
- `uint8_t lowCount2`
The low-time for carrier generator second register for FSK mode.
- `uint16_t markCount`
The mark time for the modulator gate.
- `uint16_t spaceCount`
The space time for the modulator gate.

9.4.1.0.0.12 Field Documentation

9.4.1.0.0.12.1 `uint8_t cmt_modulate_config_t::highCount1`

9.4.1.0.0.12.2 `uint8_t cmt_modulate_config_t::lowCount1`

9.4.1.0.0.12.3 `uint8_t cmt_modulate_config_t::highCount2`

9.4.1.0.0.12.4 `uint8_t cmt_modulate_config_t::lowCount2`

9.4.1.0.0.12.5 `uint16_t cmt_modulate_config_t::markCount`

9.4.1.0.0.12.6 `uint16_t cmt_modulate_config_t::spaceCount`

9.4.2 struct `cmt_config_t`

Data Fields

- `bool isInterruptEnabled`
Timer interrupt 0-disable, 1-enable.
- `bool isIroEnabled`
The IRO output 0-disabled, 1-enabled.
- `cmt_infrared_output_polarity_t iroPolarity`
The IRO polarity.
- `cmt_second_clkdiv_t divider`
The CMT clock divide prescaler.

9.4.2.0.0.13 Field Documentation

9.4.2.0.0.13.1 `bool cmt_config_t::isInterruptEnabled`

9.4.2.0.0.13.2 `bool cmt_config_t::isIroEnabled`

9.4.2.0.0.13.3 `cmt_infrared_output_polarity_t cmt_config_t::iroPolarity`

9.4.2.0.0.13.4 `cmt_second_clkdiv_t cmt_config_t::divider`

Macro Definition Documentation

9.5.1 `#define FSL_CMT_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))`

Enumeration Type Documentation

9.6.1 `enum cmt_mode_t`

Enumerator

kCMT_DirectIROCtl Carrier modulator is disabled and the IRO signal is directly in software control.

kCMT_TimeMode Carrier modulator is enabled in time mode.

kCMT_FSKMode Carrier modulator is enabled in FSK mode.

kCMT_BasebandMode Carrier modulator is enabled in baseband mode.

9.6.2 `enum cmt_primary_clkdiv_t`

The primary clock divider is used to divider the bus clock to get the intermediate frequency to approximately equal to 8 MHZ. When the bus clock is 8 MHZ, set primary prescaler to "kCMT_PrimaryClkDiv1".

Enumerator

kCMT_PrimaryClkDiv1 The intermediate frequency is the bus clock divided by 1.

kCMT_PrimaryClkDiv2 The intermediate frequency is the bus clock divided by 2.

kCMT_PrimaryClkDiv3 The intermediate frequency is the bus clock divided by 3.

kCMT_PrimaryClkDiv4 The intermediate frequency is the bus clock divided by 4.

kCMT_PrimaryClkDiv5 The intermediate frequency is the bus clock divided by 5.

kCMT_PrimaryClkDiv6 The intermediate frequency is the bus clock divided by 6.

kCMT_PrimaryClkDiv7 The intermediate frequency is the bus clock divided by 7.

kCMT_PrimaryClkDiv8 The intermediate frequency is the bus clock divided by 8.

kCMT_PrimaryClkDiv9 The intermediate frequency is the bus clock divided by 9.

kCMT_PrimaryClkDiv10 The intermediate frequency is the bus clock divided by 10.

kCMT_PrimaryClkDiv11 The intermediate frequency is the bus clock divided by 11.

kCMT_PrimaryClkDiv12 The intermediate frequency is the bus clock divided by 12.

kCMT_PrimaryClkDiv13 The intermediate frequency is the bus clock divided by 13.

kCMT_PrimaryClkDiv14 The intermediate frequency is the bus clock divided by 14.

kCMT_PrimaryClkDiv15 The intermediate frequency is the bus clock divided by 15.

kCMT_PrimaryClkDiv16 The intermediate frequency is the bus clock divided by 16.

9.6.3 `enum cmt_second_clkdiv_t`

The second prescaler can be used to divide the 8 MHZ CMT clock by 1, 2, 4, or 8 according to the specification.

Function Documentation

Enumerator

- kCMT_SecondClkDiv1* The CMT clock is the intermediate frequency frequency divided by 1.
- kCMT_SecondClkDiv2* The CMT clock is the intermediate frequency frequency divided by 2.
- kCMT_SecondClkDiv4* The CMT clock is the intermediate frequency frequency divided by 4.
- kCMT_SecondClkDiv8* The CMT clock is the intermediate frequency frequency divided by 8.

9.6.4 enum cmt_infrared_output_polarity_t

Enumerator

- kCMT_IROActiveLow* The CMT infrared output signal polarity is active-low.
- kCMT_IROActiveHigh* The CMT infrared output signal polarity is active-high.

9.6.5 enum cmt_infrared_output_state_t

Enumerator

- kCMT_IROCtlLow* The CMT Infrared output signal state is controlled to low.
- kCMT_IROCtlHigh* The CMT Infrared output signal state is controlled to high.

9.6.6 enum _cmt_interrupt_enable

This structure contains the settings for all of the CMT interrupt configurations.

Enumerator

- kCMT_EndOfCycleInterruptEnable* CMT end of cycle interrupt.

Function Documentation

9.7.1 void CMT_GetDefaultConfig (cmt_config_t * *config*)

This API gets the default configuration structure for the [CMT_Init\(\)](#). Use the initialized structure unchanged in [CMT_Init\(\)](#) or modify fields of the structure before calling the [CMT_Init\(\)](#).

Parameters

<i>config</i>	The CMT configuration structure pointer.
---------------	--

9.7.2 void CMT_Init (CMT_Type * *base*, const cmt_config_t * *config*, uint32_t *busClock_Hz*)

This function ungates the module clock and sets the CMT internal clock, interrupt, and infrared output signal for the CMT module.

Parameters

<i>base</i>	CMT peripheral base address.
<i>config</i>	The CMT basic configuration structure.
<i>busClock_Hz</i>	The CMT module input clock - bus clock frequency.

9.7.3 void CMT_Deinit (CMT_Type * *base*)

This function disables CMT modulator, interrupts, and gates the CMT clock control. CMT_Init must be called to use the CMT again.

Parameters

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

9.7.4 void CMT_SetMode (CMT_Type * *base*, cmt_mode_t *mode*, cmt_modulate_config_t * *modulateConfig*)

Parameters

<i>base</i>	CMT peripheral base address.
<i>mode</i>	The CMT feature mode enumeration. See "cmt_mode_t".
<i>modulate-Config</i>	The carrier generation and modulator configuration.

9.7.5 cmt_mode_t CMT_GetMode (CMT_Type * *base*)

Function Documentation

Parameters

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

Returns

The CMT mode. kCMT_DirectIROCtl Carrier modulator is disabled; the IRO signal is directly in software control. kCMT_TimeMode Carrier modulator is enabled in time mode. kCMT_FSKMode Carrier modulator is enabled in FSK mode. kCMT_BasebandMode Carrier modulator is enabled in baseband mode.

9.7.6 `uint32_t CMT_GetCMTFrequency (CMT_Type * base, uint32_t busClock_Hz)`

Parameters

<i>base</i>	CMT peripheral base address.
<i>busClock_Hz</i>	CMT module input clock - bus clock frequency.

Returns

The CMT clock frequency.

9.7.7 `static void CMT_SetCarrirGenerateCountOne (CMT_Type * base, uint32_t highCount, uint32_t lowCount) [inline], [static]`

This function sets the high-time and low-time of the primary data set for the CMT carrier generator counter to control the period and the duty cycle of the output carrier signal. If the CMT clock period is Tcmt, the period of the carrier generator signal equals (highCount + lowCount) * Tcmt. The duty cycle equals to highCount / (highCount + lowCount).

Parameters

<i>base</i>	CMT peripheral base address.
<i>highCount</i>	The number of CMT clocks for carrier generator signal high time, integer in the range of 1 ~ 0xFF.

<i>lowCount</i>	The number of CMT clocks for carrier generator signal low time, integer in the range of 1 ~ 0xFF.
-----------------	---

9.7.8 static void CMT_SetCarrirGenerateCountTwo (CMT_Type * *base*, uint32_t *highCount*, uint32_t *lowCount*) [inline], [static]

This function is used for FSK mode setting the high-time and low-time of the secondary data set CMT carrier generator counter to control the period and the duty cycle of the output carrier signal. If the CMT clock period is T_{cmt} , the period of the carrier generator signal equals $(highCount + lowCount) * T_{cmt}$. The duty cycle equals $highCount / (highCount + lowCount)$.

Parameters

<i>base</i>	CMT peripheral base address.
<i>highCount</i>	The number of CMT clocks for carrier generator signal high time, integer in the range of 1 ~ 0xFF.
<i>lowCount</i>	The number of CMT clocks for carrier generator signal low time, integer in the range of 1 ~ 0xFF.

9.7.9 void CMT_SetModulateMarkSpace (CMT_Type * *base*, uint32_t *markCount*, uint32_t *spaceCount*)

This function sets the mark time period of the CMT modulator counter to control the mark time of the output modulated signal from the carrier generator output signal. If the CMT clock frequency is F_{cmt} and the carrier out signal frequency is f_{cg} :

- In Time and Baseband mode: The mark period of the generated signal equals $(markCount + 1) / (F_{cmt}/8)$. The space period of the generated signal equals $spaceCount / (F_{cmt}/8)$.
- In FSK mode: The mark period of the generated signal equals $(markCount + 1)/f_{cg}$. The space period of the generated signal equals $spaceCount / f_{cg}$.

Parameters

<i>base</i>	Base address for current CMT instance.
<i>markCount</i>	The number of clock period for CMT modulator signal mark period, in the range of 0 ~ 0xFFFF.
<i>spaceCount</i>	The number of clock period for CMT modulator signal space period, in the range of the 0 ~ 0xFFFF.

9.7.10 static void CMT_EnableExtendedSpace (CMT_Type * *base*, bool *enable*) [inline], [static]

This function is used to make the space period longer for time, baseband, and FSK modes.

Parameters

<i>base</i>	CMT peripheral base address.
<i>enable</i>	True enable the extended space, false disable the extended space.

9.7.11 void CMT_SetIroState (CMT_Type * *base*, cmt_infrared_output_state_t *state*)

Changes the states of the IRO signal when the kCMT_DirectIROMode mode is set and the IRO signal is enabled.

Parameters

<i>base</i>	CMT peripheral base address.
<i>state</i>	The control of the IRO signal. See "cmt_infrared_output_state_t"

9.7.12 static void CMT_EnableInterrupts (CMT_Type * *base*, uint32_t *mask*) [inline], [static]

This function enables the CMT interrupts according to the provided mask if enabled. The CMT only has the end of the cycle interrupt - an interrupt occurs at the end of the modulator cycle. This interrupt provides a means for the user to reload the new mark/space values into the CMT modulator data registers and verify the modulator mark and space. For example, to enable the end of cycle, do the following.

```
*  CMT_EnableInterrupts(CMT,
*  kCMT_EndOfCycleInterruptEnable);
```

Parameters

<i>base</i>	CMT peripheral base address.
<i>mask</i>	The interrupts to enable. Logical OR of _cmt_interrupt_enable .

9.7.13 static void CMT_DisableInterrupts (CMT_Type * *base*, uint32_t *mask*) [inline], [static]

This function disables the CMT interrupts according to the provided maskIf enabled. The CMT only has the end of the cycle interrupt. For example, to disable the end of cycle, do the following.

```
*  CMT_DisableInterrupts(CMT,
*  kCMT_EndOfCycleInterruptEnable);
```

Function Documentation

Parameters

<i>base</i>	CMT peripheral base address.
<i>mask</i>	The interrupts to enable. Logical OR of _cmt_interrupt_enable .

9.7.14 `static uint32_t CMT_GetStatusFlags (CMT_Type * base) [inline], [static]`

The flag is set:

- When the modulator is not currently active and carrier and modulator are set to start the initial CMT transmission.
- At the end of each modulation cycle when the counter is reloaded and the carrier and modulator are enabled.

Parameters

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

Returns

Current status of the end of cycle status flag

- non-zero: End-of-cycle has occurred.
- zero: End-of-cycle has not yet occurred since the flag last cleared.

Chapter 10 Common Driver

Overview

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

Macros

- #define **MAKE_STATUS**(group, code) (((group)*100) + (code))
Construct a status code value from a group and code number.
- #define **MAKE_VERSION**(major, minor, bugfix) (((major) << 16) | ((minor) << 8) | (bugfix))
Construct the version number for drivers.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_NONE** 0U
No debug console.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_UART** 1U
Debug console based on UART.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_LPUART** 2U
Debug console based on LPUART.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_LPSCI** 3U
Debug console based on LPSCI.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_USBCDC** 4U
Debug console based on USBCDC.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM** 5U
Debug console based on FLEXCOMM.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_IUART** 6U
Debug console based on i.MX UART.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_VUSART** 7U
Debug console based on LPC_VUSART.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART** 8U
Debug console based on LPC_USART.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_SWO** 9U
Debug console based on SWO.
- #define **ARRAY_SIZE**(x) (sizeof(x) / sizeof((x)[0]))
Computes the number of elements in an array.

Typedefs

- typedef int32_t **status_t**
Type used for all status and error return values.

Enumerations

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


```
kStatusGroup_SDIO_SLV = 151 }
```

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 = MAKE_STATUS(kStatusGroup_Generic, 6) }

Generic status return codes.

Functions

- static [status_t EnableIRQ](#) (IRQn_Type interrupt)
 Enable specific interrupt.
- static [status_t DisableIRQ](#) (IRQn_Type interrupt)
 Disable specific interrupt.
- static uint32_t [DisableGlobalIRQ](#) (void)
 Disable the global IRQ.
- static void [EnableGlobalIRQ](#) (uint32_t primask)
 Enable the global IRQ.
- void * [SDK_Malloc](#) (size_t size, size_t alignbytes)
 Allocate memory with given alignment and aligned size.
- void [SDK_Free](#) (void *ptr)
 Free memory.
- void [SDK_DelayAtLeastUs](#) (uint32_t delay_us, uint32_t coreClock_Hz)
 Delay at least for some time.

Driver version

- #define [FSL_COMMON_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 2, 4))
 common driver version 2.2.4.

Min/max macros

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

UINT16_MAX/UINT32_MAX value

- #define [UINT16_MAX](#) ((uint16_t)-1)
- #define [UINT32_MAX](#) ((uint32_t)-1)

Timer utilities

- #define [USEC_TO_COUNT](#)(us, clockFreqInHz) (uint64_t)((((uint64_t)(us) * (clockFreqInHz)) / 1000000U))
 Macro to convert a microsecond period to raw count value.

Overview

- #define **COUNT_TO_USEC**(count, clockFreqInHz) (uint64_t)((uint64_t)(count) * 1000000U / (clockFreqInHz))
Macro to convert a raw count value to microsecond.
- #define **MSEC_TO_COUNT**(ms, clockFreqInHz) (uint64_t)((uint64_t)(ms) * (clockFreqInHz) / 1000U)
Macro to convert a millisecond period to raw count value.
- #define **COUNT_TO_MSEC**(count, clockFreqInHz) (uint64_t)((uint64_t)(count) * 1000U / (clockFreqInHz))
Macro to convert a raw count value to millisecond.

Alignment variable definition macros

- #define **SDK_ALIGN**(var, alignbytes) var
- #define **SDK_SIZEALIGN**(var, alignbytes) ((unsigned int)((var) + ((alignbytes)-1U)) & (unsigned int)(~(unsigned int)((alignbytes)-1U)))
Macro to change a value to a given size aligned value.

Non-cacheable region definition macros

- #define **AT_NONCACHEABLE_SECTION**(var) var
- #define **AT_NONCACHEABLE_SECTION_ALIGN**(var, alignbytes) var
- #define **AT_NONCACHEABLE_SECTION_INIT**(var) var
- #define **AT_NONCACHEABLE_SECTION_ALIGN_INIT**(var, alignbytes) var

Suppress fallthrough warning macro

- #define **SUPPRESS_FALL_THROUGH_WARNING**()

Macro Definition Documentation

10.2.1 **#define MAKE_STATUS(*group*, *code*)** ((((*group*)*100) + (*code*)))

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

10.2.3 **#define FSL_COMMON_DRIVER_VERSION** (MAKE_VERSION(2, 2, 4))

10.2.4 **#define DEBUG_CONSOLE_DEVICE_TYPE_NONE** 0U

10.2.5 **#define DEBUG_CONSOLE_DEVICE_TYPE_UART** 1U

10.2.6 **#define DEBUG_CONSOLE_DEVICE_TYPE_LPUART** 2U

10.2.7 **#define DEBUG_CONSOLE_DEVICE_TYPE_LPSCI** 3U

10.2.8 **#define DEBUG_CONSOLE_DEVICE_TYPE_USBCDC** 4U

10.2.9 **#define DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM** 5U

10.2.10 **#define DEBUG_CONSOLE_DEVICE_TYPE_IUART** 6U

10.2.11 **#define DEBUG_CONSOLE_DEVICE_TYPE_VUSART** 7U

10.2.12 **#define DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART** 8U

10.2.13 **#define DEBUG_CONSOLE_DEVICE_TYPE_SWO** 9U

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

Typedef Documentation

10.3.1 **typedef int32_t** status_t

Enumeration Type Documentation

10.4.1 **enum** _status_groups

Enumerator

kStatusGroup_Generic Group number for generic status codes.

kStatusGroup_FLASH Group number for FLASH status codes.

kStatusGroup_LPSPI Group number for LPSPI status codes.

kStatusGroup_FLEXIO_SPI Group number for FLEXIO SPI status codes.

kStatusGroup_DSPI Group number for DSPI status codes.

kStatusGroup_FLEXIO_UART Group number for FLEXIO UART status codes.

kStatusGroup_FLEXIO_I2C Group number for FLEXIO I2C status codes.

kStatusGroup_LPI2C Group number for LPI2C status codes.

kStatusGroup_UART Group number for UART status codes.

kStatusGroup_I2C Group number for 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_SDSPI Group number for SDSPI status codes.

kStatusGroup_FLEXIO_I2S Group number for FLEXIO I2S status codes.

kStatusGroup_FLEXIO_MCULCD Group number for FLEXIO LCD status codes.

kStatusGroup_FLASHIAP Group number for FLASHIAP status codes.

kStatusGroup_FLEXCOMM_I2C Group number for FLEXCOMM I2C status codes.

kStatusGroup_I2S Group number for I2S status codes.

kStatusGroup_IUART Group number for IUART status codes.

kStatusGroup_CSI Group number for CSI status codes.

kStatusGroup_MIPI_DSI Group number for MIPI DSI status codes.

kStatusGroup_SDRAMC Group number for SDRAMC status codes.

kStatusGroup_POWER Group number for POWER status codes.

kStatusGroup_ENET Group number for ENET status codes.

kStatusGroup_PHY Group number for PHY status codes.

kStatusGroup_TRGMUX Group number for TRGMUX status codes.

kStatusGroup_SMARTCARD Group number for SMARTCARD status codes.

kStatusGroup_LMEM Group number for LMEM status codes.

kStatusGroup_QSPI Group number for QSPI status codes.

kStatusGroup_DMA Group number for DMA status codes.

kStatusGroup_EDMA Group number for EDMA status codes.

kStatusGroup_DMAMGR Group number for DMAMGR status codes.

kStatusGroup_FLEXCAN Group number for FlexCAN status codes.

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_HAL_GPIO Group number for HAL GPIO status codes.
kStatusGroup_HAL_UART Group number for HAL UART status codes.
kStatusGroup_HAL_TIMER Group number for HAL TIMER status codes.
kStatusGroup_HAL_SPI Group number for HAL SPI status codes.
kStatusGroup_HAL_I2C Group number for HAL I2C status codes.
kStatusGroup_HAL_FLASH Group number for HAL FLASH status codes.
kStatusGroup_HAL_PWM Group number for HAL PWM status codes.
kStatusGroup_HAL_RNG Group number for HAL RNG status codes.
kStatusGroup_TIMERMANAGER Group number for TiMER MANAGER status codes.
kStatusGroup_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.

Function Documentation

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_CODEEC Group number for codec status codes.
kStatusGroup_ASRC Group number for codec status ASRC.
kStatusGroup_OTFAD Group number for codec status codes.
kStatusGroup_SDIOISLV Group number for SDIOISLV status codes.

10.4.2 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.

Function Documentation

10.5.1 static status_t EnableIRQ (IRQn_Type *interrupt*) [inline], [static]

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

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

Parameters

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

Return values

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

10.5.2 static status_t DisableIRQ (IRQn_Type *interrupt*) [inline], [static]

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

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

Parameters

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

Return values

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

10.5.3 static uint32_t DisableGlobalIRQ (void) [inline], [static]

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

Returns

Current primask value.

10.5.4 static void EnableGlobalIRQ (uint32_t *primask*) [inline], [static]

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

Function Documentation

Parameters

<i>primask</i>	value of primask register to be restored. The primask value is supposed to be provided by the DisableGlobalIRQ() .
----------------	--

10.5.5 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.
------------	-------------------

10.5.6 void SDK_Free (void * ptr)

Parameters

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

10.5.7 void SDK_DelayAtLeastUs (uint32_t delay_us, uint32_t coreClock_Hz)

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

Parameters

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

Chapter 11

CRC: Cyclic Redundancy Check Driver

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.

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.

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.

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.

Comments about API usage in RTOS

[CRC_Init\(\)](#) / [CRC_WriteData\(\)](#) / [CRC_Get16bitResult\(\)](#) to get an intermediate checksum.

[CRC_Init\(\)](#) / [CRC_WriteData\(\)](#) / ... / [CRC_WriteData\(\)](#) / [CRC_Get16bitResult\(\)](#) to get an intermediate checksum.

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/crcRefer 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/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)

- Loads default values to the CRC protocol configuration structure.
- 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, 2))
CRC driver version.

Data Structure Documentation

11.6.1 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\(\)](#)

11.6.1.0.0.14 Field Documentation

11.6.1.0.0.14.1 uint32_t crc_config_t::polynomial

Example polynomial: 0x1021 = 1_0000_0010_0001 = $x^{12} + x^5 + 1$

11.6.1.0.0.14.2 bool crc_config_t::reflectIn

11.6.1.0.0.14.3 bool crc_config_t::reflectOut

11.6.1.0.0.14.4 bool crc_config_t::complementChecksum

11.6.1.0.0.14.5 [crc_bits_t](#) crc_config_t::crcBits

Function Documentation

Macro Definition Documentation

11.7.1 #define FSL_CRC_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

Version 2.0.1.

Current version: 2.0.1

Change log:

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

11.7.2 #define CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT 1

Use CRC16-CCIT-FALSE as default.

Enumeration Type Documentation

11.8.1 enum crc_bits_t

Enumerator

kCrcBits16 Generate 16-bit CRC code.

kCrcBits32 Generate 32-bit CRC code.

11.8.2 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.

Function Documentation

11.9.1 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.

11.9.2 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.
-------------	-------------------------

11.9.3 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.
---------------	---------------------------------------

11.9.4 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

Function Documentation

<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.

11.9.5 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.

11.9.6 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 12

DAC: Digital-to-Analog Converter Driver

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.

Typical use case

12.2.1 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`

12.2.2 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,

Typical use case

- `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 }`
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.

Data Structure Documentation

12.3.1 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.

12.3.1.0.0.15 Field Documentation

12.3.1.0.0.15.1 [dac_reference_voltage_source_t](#) [dac_config_t::referenceVoltageSource](#)

12.3.1.0.0.15.2 [bool](#) [dac_config_t::enableLowPowerMode](#)

12.3.2 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](#)

Enumeration Type Documentation

Set the upper limit for the buffer index.

12.3.2.0.0.16 Field Documentation

12.3.2.0.0.16.1 `dac_buffer_trigger_mode_t` `dac_buffer_config_t::triggerMode`

12.3.2.0.0.16.2 `dac_buffer_watermark_t` `dac_buffer_config_t::watermark`

12.3.2.0.0.16.3 `dac_buffer_work_mode_t` `dac_buffer_config_t::workMode`

12.3.2.0.0.16.4 `uint8_t` `dac_buffer_config_t::upperLimit`

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

Macro Definition Documentation

12.4.1 `#define FSL_DAC_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

Enumeration Type Documentation

12.5.1 `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.

12.5.2 `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.

12.5.3 `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.

12.5.4 enum dac_buffer_trigger_mode_t

Enumerator

kDAC_BufferTriggerByHardwareMode The DAC hardware trigger is selected.

kDAC_BufferTriggerBySoftwareMode The DAC software trigger is selected.

12.5.5 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.

12.5.6 enum dac_buffer_work_mode_t

Enumerator

kDAC_BufferWorkAsNormalMode Normal mode.

kDAC_BufferWorkAsSwingMode Swing mode.

kDAC_BufferWorkAsOneTimeScanMode One-Time Scan mode.

Function Documentation

12.6.1 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".

12.6.2 void DAC_Deinit (DAC_Type * *base*)

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

Function Documentation

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

Parameters

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

12.6.3 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".
---------------	---

12.6.4 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.

12.6.5 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.
---------------	----------------------------------

12.6.6 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".

12.6.7 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".
---------------	--

12.6.8 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.

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

Function Documentation

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.

12.6.10 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.
-------------	------------------------------

12.6.11 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.

12.6.12 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.

12.6.13 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".

12.6.14 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".

12.6.15 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".

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

Function Documentation

Parameters

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

Chapter 13

DMAMUX: Direct Memory Access Multiplexer Driver

Overview

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

Typical use case

13.2.1 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, 4))
DMAMUX driver version 2.0.4.

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.

Macro Definition Documentation

13.3.1 #define FSL_DMAMUX_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

Function Documentation

13.4.1 void DMAMUX_Init (DMAMUX_Type * *base*)

This function ungates the DMAMUX clock.

Parameters

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

13.4.2 void DMAMUX_Deinit (DMAMUX_Type * *base*)

This function gates the DMAMUX clock.

Parameters

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

13.4.3 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.

13.4.4 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.
-------------	---------------------------------

Function Documentation

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

13.4.5 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.

13.4.6 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.

13.4.7 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 14

DSPI: Serial Peripheral Interface Driver

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](#)

DSPI Driver

DSPI Driver

14.2.1 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.

14.2.2 Typical use case

14.2.2.1 Master Operation

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

14.2.2.2 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 data 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_PcsToSck` = 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` []

DSPI Driver

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 src-Clock_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 PUSHHR data register address for the DMA operation.*
static uint32_t [DSPI_SlaveGetTxRegisterAddress](#) (SPI_Type *base)
- *Gets the DSPI slave PUSHHR 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 [DSPI_GetInstance](#) (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 PUSHHR 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)

DSPI Driver

- 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.

14.2.3 Data Structure Documentation

14.2.3.1 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.

14.2.3.1.0.17 Field Documentation

14.2.3.1.0.17.1 bool dspi_command_data_config_t::isPcsContinuous

14.2.3.1.0.17.2 uint8_t dspi_command_data_config_t::whichCtar

14.2.3.1.0.17.3 uint8_t dspi_command_data_config_t::whichPcs

14.2.3.1.0.17.4 bool dspi_command_data_config_t::isEndOfQueue

14.2.3.1.0.17.5 bool dspi_command_data_config_t::clearTransferCount

14.2.3.2 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.

DSPI Driver

14.2.3.2.0.18 Field Documentation

14.2.3.2.0.18.1 `uint32_t dspi_master_ctar_config_t::baudRate`

14.2.3.2.0.18.2 `uint32_t dspi_master_ctar_config_t::bitsPerFrame`

14.2.3.2.0.18.3 `dspi_clock_polarity_t dspi_master_ctar_config_t::cpol`

14.2.3.2.0.18.4 `dspi_clock_phase_t dspi_master_ctar_config_t::cpha`

14.2.3.2.0.18.5 `dspi_shift_direction_t dspi_master_ctar_config_t::direction`

14.2.3.2.0.18.6 `uint32_t dspi_master_ctar_config_t::pcsToSckDelayInNanoSec`

It also sets the boundary value if out of range.

14.2.3.2.0.18.7 `uint32_t dspi_master_ctar_config_t::lastSckToPcsDelayInNanoSec`

It also sets the boundary value if out of range.

14.2.3.2.0.18.8 `uint32_t dspi_master_ctar_config_t::betweenTransferDelayInNanoSec`

It also sets the boundary value if out of range.

14.2.3.3 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.

14.2.3.3.0.19 Field Documentation

14.2.3.3.0.19.1 `dspi_ctar_selection_t dspi_master_config_t::whichCtar`

14.2.3.3.0.19.2 `dspi_master_ctar_config_t dspi_master_config_t::ctarConfig`

14.2.3.3.0.19.3 `dspi_which_pcs_t dspi_master_config_t::whichPcs`

14.2.3.3.0.19.4 `dspi_pcs_polarity_config_t dspi_master_config_t::pcsActiveHighOrLow`

14.2.3.3.0.19.5 `bool dspi_master_config_t::enableContinuousSCK`

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

14.2.3.3.0.19.6 `bool dspi_master_config_t::enableRxFifoOverWrite`

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.

14.2.3.3.0.19.7 `bool dspi_master_config_t::enableModifiedTimingFormat`

14.2.3.3.0.19.8 `dspi_master_sample_point_t dspi_master_config_t::samplePoint`

It's valid only when CPHA=0.

14.2.3.4 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.

14.2.3.4.0.20 Field Documentation

14.2.3.4.0.20.1 `uint32_t dspi_slave_ctar_config_t::bitsPerFrame`

14.2.3.4.0.20.2 `dspi_clock_polarity_t dspi_slave_ctar_config_t::cpol`

14.2.3.4.0.20.3 `dspi_clock_phase_t dspi_slave_ctar_config_t::cpha`

Slave only supports MSB and does not support LSB.

14.2.3.5 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` [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.

14.2.3.5.0.21 Field Documentation

14.2.3.5.0.21.1 `dspi_ctar_selection_t dspi_slave_config_t::whichCtar`

14.2.3.5.0.21.2 `dspi_slave_ctar_config_t dspi_slave_config_t::ctarConfig`

14.2.3.5.0.21.3 `bool dspi_slave_config_t::enableContinuousSCK`

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

14.2.3.5.0.21.4 `bool dspi_slave_config_t::enableRxFifoOverWrite`

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.

14.2.3.5.0.21.5 `bool dspi_slave_config_t::enableModifiedTimingFormat`

14.2.3.5.0.21.6 `dspi_master_sample_point_t dspi_slave_config_t::samplePoint`

It's valid only when CPHA=0.

14.2.3.6 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.

14.2.3.6.0.22 Field Documentation

14.2.3.6.0.22.1 `uint8_t* dspi_transfer_t::txData`

14.2.3.6.0.22.2 `uint8_t* dspi_transfer_t::rxData`

14.2.3.6.0.22.3 `volatile size_t dspi_transfer_t::dataSize`

14.2.3.6.0.22.4 `uint32_t dspi_transfer_t::configFlags`

Set from `_dspi_transfer_config_flag_for_master` if the transfer is used for master or `_dspi_transfer_config_flag_for_slave` enumeration if the transfer is used for slave.

14.2.3.7 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 `_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.

14.2.3.7.0.23 Field Documentation

14.2.3.7.0.23.1 `uint32_t dspi_half_duplex_transfer_t::configFlags`

14.2.3.7.0.23.2 `bool dspi_half_duplex_transfer_t::isPcsAssertInTransfer`

true for assert and false for de-assert.

14.2.3.7.0.23.3 `bool dspi_half_duplex_transfer_t::isTransmitFirst`

14.2.3.8 struct _dspi_master_handle

Forward declaration of the `_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](#)
The desired last data command.
- 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 [_dsapi_transfer_state](#).
- [dsapi_master_transfer_callback_t](#) callback
Completion callback.
- void * [userData](#)
Callback user data.

14.2.3.8.0.24 Field Documentation

- 14.2.3.8.0.24.1 `uint32_t dspi_master_handle_t::bitsPerFrame`
- 14.2.3.8.0.24.2 `volatile uint32_t dspi_master_handle_t::command`
- 14.2.3.8.0.24.3 `volatile uint32_t dspi_master_handle_t::lastCommand`
- 14.2.3.8.0.24.4 `uint8_t dspi_master_handle_t::fifoSize`
- 14.2.3.8.0.24.5 `volatile bool dspi_master_handle_t::isPcsActiveAfterTransfer`
- 14.2.3.8.0.24.6 `volatile bool dspi_master_handle_t::isThereExtraByte`
- 14.2.3.8.0.24.7 `uint8_t* volatile dspi_master_handle_t::txData`
- 14.2.3.8.0.24.8 `uint8_t* volatile dspi_master_handle_t::rxData`
- 14.2.3.8.0.24.9 `volatile size_t dspi_master_handle_t::remainingSendByteCount`
- 14.2.3.8.0.24.10 `volatile size_t dspi_master_handle_t::remainingReceiveByteCount`
- 14.2.3.8.0.24.11 `volatile uint8_t dspi_master_handle_t::state`
- 14.2.3.8.0.24.12 `dspi_master_transfer_callback_t dspi_master_handle_t::callback`
- 14.2.3.8.0.24.13 `void* dspi_master_handle_t::userData`

14.2.3.9 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 Driver

- 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.

14.2.3.9.0.25 Field Documentation

- 14.2.3.9.0.25.1 uint32_t dspi_slave_handle_t::bitsPerFrame
- 14.2.3.9.0.25.2 volatile bool dspi_slave_handle_t::isThereExtraByte
- 14.2.3.9.0.25.3 uint8_t* volatile dspi_slave_handle_t::txData
- 14.2.3.9.0.25.4 uint8_t* volatile dspi_slave_handle_t::rxData
- 14.2.3.9.0.25.5 volatile size_t dspi_slave_handle_t::remainingSendByteCount
- 14.2.3.9.0.25.6 volatile size_t dspi_slave_handle_t::remainingReceiveByteCount
- 14.2.3.9.0.25.7 volatile uint8_t dspi_slave_handle_t::state
- 14.2.3.9.0.25.8 volatile uint32_t dspi_slave_handle_t::errorCount
- 14.2.3.9.0.25.9 dspi_slave_transfer_callback_t dspi_slave_handle_t::callback
- 14.2.3.9.0.25.10 void* dspi_slave_handle_t::userData

14.2.4 Macro Definition Documentation

- 14.2.4.1 #define FSL_DSPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 4))
- 14.2.4.2 #define DSPI_DUMMY_DATA (0x00U)

Dummy data used for Tx if there is no txData.

14.2.4.3 `#define DSPI_MASTER_CTAR_SHIFT (0U)`

14.2.4.4 `#define DSPI_MASTER_CTAR_MASK (0x0FU)`

14.2.4.5 `#define DSPI_MASTER_PCS_SHIFT (4U)`

14.2.4.6 `#define DSPI_MASTER_PCS_MASK (0xF0U)`

14.2.4.7 `#define DSPI_SLAVE_CTAR_SHIFT (0U)`

14.2.4.8 `#define DSPI_SLAVE_CTAR_MASK (0x07U)`

14.2.5 Typedef Documentation

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

DSPI Driver

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.

14.2.5.2 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.

14.2.6 Enumeration Type Documentation

14.2.6.1 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.

14.2.6.2 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.

14.2.6.3 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.

14.2.6.4 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.

14.2.6.5 enum dsapi_master_slave_mode_t

Enumerator

kDSPI_Master DSPI peripheral operates in master mode.
kDSPI_Slave DSPI peripheral operates in slave mode.

14.2.6.6 enum dsapi_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.

14.2.6.7 enum dsapi_which_pcs_t

Enumerator

kDSPI_Pcs0 Pcs[0].
kDSPI_Pcs1 Pcs[1].
kDSPI_Pcs2 Pcs[2].

DSPI Driver

kDSPI_Pcs3 Pcs[3].

kDSPI_Pcs4 Pcs[4].

kDSPI_Pcs5 Pcs[5].

14.2.6.8 enum dspi_pcs_polarity_config_t

Enumerator

kDSPI_PcsActiveHigh Pcs Active High (idles low).

kDSPI_PcsActiveLow Pcs Active Low (idles high).

14.2.6.9 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).

14.2.6.10 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).

14.2.6.11 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.

14.2.6.12 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**14.2.6.13 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.**14.2.6.14 enum dspi_ctar_selection_t**

Enumerator

kDSPI_Ctar0 CTAR0 selection option for master or slave mode; note that CTAR0 and CTAR0_SLAVE 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.**14.2.6.15 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.

DSPI Driver

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.

14.2.6.16 enum _dspi_transfer_config_flag_for_slave

Enumerator

kDSPI_SlaveCtar0 DSPI slave transfer use CTAR0 setting. DSPI slave can only use PCS0.

14.2.6.17 enum _dspi_transfer_state

Enumerator

kDSPI_Idle Nothing in the transmitter/receiver.
kDSPI_Busy Transfer queue is not finished.
kDSPI_Error Transfer error.

14.2.7 Function Documentation

14.2.7.1 void DSPI_MasterInit (SPI_Type * base, const dspi_master_config_t * masterConfig, uint32_t srcClock_Hz)

This function initializes the DSPI master configuration. This is an example use case.

```
* dspi_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      = 1000000000U /
    masterConfig.ctarConfig.baudRate ;
*   masterConfig.ctarConfig.lastSckToPcsDelayInNanoSec  = 1000000000U
    / masterConfig.ctarConfig.baudRate ;
*   masterConfig.ctarConfig.betweenTransferDelayInNanoSec =
    1000000000U / masterConfig.ctarConfig.baudRate ;
*   masterConfig.whichPcs                               = kDSPI_Pcs0;
*   masterConfig.pcsActiveHighOrLow                     =
    kDSPI_PcsActiveLow;
*   masterConfig.enableContinuousSCK                   = false;
*   masterConfig.enableRxFifoOverWrite                 = false;
*   masterConfig.enableModifiedTimingFormat            = false;
*   masterConfig.samplePoint                           =
    kDSPI_SckToSin0Clock;
*   DSPI_MasterInit(base, &masterConfig, srcClock_Hz);
*

```

Parameters

<i>base</i>	DSPI peripheral address.
<i>masterConfig</i>	Pointer to the structure dspi_master_config_t .
<i>srcClock_Hz</i>	Module source input clock in Hertz.

14.2.7.2 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 dspi_master_config_t structure
---------------------	---

14.2.7.3 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->ctarConfig.bitsPerFrame  = 8;
*   slaveConfig->ctarConfig.cpol          =
    kDSPI_ClockPolarityActiveHigh;
*   slaveConfig->ctarConfig.cpha          =
    kDSPI_ClockPhaseFirstEdge;
*   slaveConfig->enableContinuousSCK      = false;

```

DSPI Driver

```
* slaveConfig->enableRxFifoOverWrite    = 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 dspi_master_config_t .

14.2.7.4 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 dspi_slave_config_t structure.
--------------------	---

14.2.7.5 void DSPI_Deinit (SPI_Type * *base*)

Call this API to disable the DSPI clock.

Parameters

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

14.2.7.6 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.

14.2.7.7 static uint32_t DSPI_GetStatusFlags (SPI_Type * *base*) [inline], [static]

Parameters

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

Returns

DSPI status (in SR register).

14.2.7.8 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 dspi_flags.

< The status flags are cleared by writing 1 (w1c).

14.2.7.9 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 );
*
```

DSPI Driver

Parameters

<i>base</i>	DSPI peripheral address.
<i>mask</i>	The interrupt mask; use the enum _dspi_interrupt_enable .

14.2.7.10 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 _dspi_interrupt_enable .

14.2.7.11 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 _dspi_dma_enable .

14.2.7.12 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.

```
* DSPI_DisableDMA(base, kDSPI_TxDmaEnable | kDSPI_RxDmaEnable);  
*
```

Parameters

<i>base</i>	DSPI peripheral address.
<i>mask</i>	The interrupt mask; use the enum _dspi_dma_enable .

14.2.7.13 **static uint32_t DSPI_MasterGetTxRegisterAddress (SPI_Type * *base*) [inline], [static]**

This function gets the DSPI master PUSHHR data register address because this value is needed for the DMA operation.

Parameters

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

Returns

The DSPI master PUSHHR data register address.

14.2.7.14 **static uint32_t DSPI_SlaveGetTxRegisterAddress (SPI_Type * *base*) [inline], [static]**

This function gets the DSPI slave PUSHHR data register address as this value is needed for the DMA operation.

Parameters

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

Returns

The DSPI slave PUSHHR data register address.

14.2.7.15 **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.

DSPI Driver

Parameters

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

Returns

The DSPI POPR data register address.

14.2.7.16 `uint32_t DSPI_GetInstance (SPI_Type * base)`

Parameters

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

14.2.7.17 `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 dspi_master_slave_mode_t .

14.2.7.18 `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.

14.2.7.19 `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.
-------------	--------------------------

14.2.7.20 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.
-------------	--------------------------

14.2.7.21 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

14.2.7.22 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

DSPI Driver

<i>flushRxFifo</i>	Flushes (true) the Rx FIFO; Otherwise, does not flush (false) the Rx FIFO
--------------------	---

14.2.7.23 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 _dspi_pcs_polarity .

14.2.7.24 uint32_t DSPI_MasterSetBaudRate (SPI_Type * *base*, dspi_ctar_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 dspi_ctar_selection_t
<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

14.2.7.25 void DSPI_MasterSetDelayScaler (SPI_Type * *base*, dspi_ctar_selection_t *whichCtar*, uint32_t *prescaler*, uint32_t *scaler*, dspi_delay_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 dspi_ctar_selection_t .
<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 dspi_delay_type_t

14.2.7.26 **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 dspi_ctar_selection_t .
<i>whichDelay</i>	The desired delay to configure, must be of type dspi_delay_type_t
<i>srcClock_Hz</i>	Module source input clock in Hertz

DSPI Driver

<i>delayTimeInNanoSec</i>	The desired delay value in nanoseconds.
---------------------------	---

Returns

The actual calculated delay value.

14.2.7.27 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.

14.2.7.28 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 dspi_command_data_config_t structure.
----------------	--

14.2.7.29 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.

14.2.7.30 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 PUSH 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_WriteCommandDataMastermode** or **DSPI_HAL_WriteCommandDataMastermodeBlocking** to write the entire 32-bit command data word to the PUSH register. 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

DSPI Driver

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 PUSHHR data register bit field.

14.2.7.31 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.

14.2.7.32 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.

14.2.7.33 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.

14.2.7.34 static uint32_t DSPI_ReadData (SPI_Type * *base*) [inline], [static]

Parameters

DSPI Driver

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

Returns

The data from the read data buffer.

14.2.7.35 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.

14.2.7.36 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 _dspi_master_handle .
<i>callback</i>	DSPI callback.
<i>userData</i>	Callback function parameter.

14.2.7.37 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 dspi_transfer_t structure.

Returns

status of status_t.

14.2.7.38 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 _dspi_master_handle structure which stores the transfer state.
<i>transfer</i>	Pointer to the dspi_transfer_t structure.

Returns

status of status_t.

14.2.7.39 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 dspi_half_duplex_transfer_t structure

Returns

status of status_t.

14.2.7.40 `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 _dspi_master_handle structure which stores the transfer state
<i>xfer</i>	pointer to dspi_half_duplex_transfer_t structure

Returns

status of status_t.

14.2.7.41 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 _dspi_master_handle 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.

14.2.7.42 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 _dspi_master_handle structure which stores the transfer state.

14.2.7.43 void DSPI_MasterTransferHandleIRQ (SPI_Type * *base*, dspi_master_handle_t * *handle*)

This function processes the DSPI transmit and receive IRQ.

DSPI Driver

Parameters

<i>base</i>	DSPI peripheral base address.
<i>handle</i>	Pointer to the _dspi_master_handle structure which stores the transfer state.

14.2.7.44 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 _dspi_slave_handle .
<i>base</i>	DSPI peripheral base address.
<i>callback</i>	DSPI callback.
<i>userData</i>	Callback function parameter.

14.2.7.45 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 _dspi_slave_handle structure which stores the transfer state.
<i>transfer</i>	Pointer to the dspi_transfer_t structure.

Returns

status of status_t.

14.2.7.46 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 _dspi_master_handle 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.

14.2.7.47 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 _dspi_slave_handle structure which stores the transfer state.

14.2.7.48 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 _dspi_slave_handle structure which stores the transfer state.

14.2.7.49 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.

14.2.8 Variable Documentation

14.2.8.1 volatile uint8_t g_dspiDummyData[]

DSPI eDMA Driver

DSPI eDMA Driver

14.3.1 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, dspi_master_edma_handle_t *handle, [status_t](#) status, void *userData)
Completion callback function pointer type.
- typedef void(* [dspi_slave_edma_transfer_callback_t](#))(SPI_Type *base, dspi_slave_edma_handle_t *handle, [status_t](#) status, void *userData)
Completion callback function pointer type.

Variables

- uint32_t [dspi_master_edma_handle_t::bitsPerFrame](#)
The desired number of bits per frame.
- volatile uint32_t [dspi_master_edma_handle_t::command](#)
The desired data command.
- volatile uint32_t [dspi_master_edma_handle_t::lastCommand](#)
The desired last data command.
- uint8_t [dspi_master_edma_handle_t::fifoSize](#)
FIFO dataSize.
- volatile bool [dspi_master_edma_handle_t::isPcsActiveAfterTransfer](#)
Indicates whether the PCS signal keeps active after the last frame transfer.
- uint8_t [dspi_master_edma_handle_t::nbytes](#)
eDMA minor byte transfer count initially configured.
- volatile uint8_t [dspi_master_edma_handle_t::state](#)
DSPI transfer state, see [_dspi_transfer_state](#).
- uint8_t *volatile [dspi_master_edma_handle_t::txData](#)

- *Send buffer.*
uint8_t *volatile dspi_master_edma_handle_t::rxData
- *Receive buffer.*
volatile size_t dspi_master_edma_handle_t::remainingSendByteCount
A number of bytes remaining to send.
- volatile size_t dspi_master_edma_handle_t::remainingReceiveByteCount
A number of bytes remaining to receive.
- size_t dspi_master_edma_handle_t::totalByteCount
A number of transfer bytes.
- uint32_t dspi_master_edma_handle_t::rxBuffIfNull
Used if there is not rxData for DMA purpose.
- uint32_t dspi_master_edma_handle_t::txBuffIfNull
Used if there is not txData for DMA purpose.
- dspi_master_edma_transfer_callback_t dspi_master_edma_handle_t::callback
Completion callback.
- void * dspi_master_edma_handle_t::userData
Callback user data.
- edma_handle_t * dspi_master_edma_handle_t::edmaRxRegToRxDataHandle
edma_handle_t handle point used for RxReg to RxData buff
- edma_handle_t * dspi_master_edma_handle_t::edmaTxDataToIntermediaryHandle
edma_handle_t handle point used for TxData to Intermediary
- edma_handle_t * dspi_master_edma_handle_t::edmaIntermediaryToTxRegHandle
edma_handle_t handle point used for Intermediary to TxReg
- edma_tcd_t dspi_master_edma_handle_t::dspiSoftwareTCD [2]
SoftwareTCD , internal used.
- uint32_t dspi_slave_edma_handle_t::bitsPerFrame
The desired number of bits per frame.
- uint8_t *volatile dspi_slave_edma_handle_t::txData
Send buffer.
- uint8_t *volatile dspi_slave_edma_handle_t::rxData
Receive buffer.
- volatile size_t dspi_slave_edma_handle_t::remainingSendByteCount
A number of bytes remaining to send.
- volatile size_t dspi_slave_edma_handle_t::remainingReceiveByteCount
A number of bytes remaining to receive.
- size_t dspi_slave_edma_handle_t::totalByteCount
A number of transfer bytes.
- uint32_t dspi_slave_edma_handle_t::rxBuffIfNull
Used if there is not rxData for DMA purpose.
- uint32_t dspi_slave_edma_handle_t::txBuffIfNull
Used if there is not txData for DMA purpose.
- uint32_t dspi_slave_edma_handle_t::txLastData
Used if there is an extra byte when 16bits per frame for DMA purpose.
- uint8_t dspi_slave_edma_handle_t::nbytes
eDMA minor byte transfer count initially configured.
- volatile uint8_t dspi_slave_edma_handle_t::state
DSPI transfer state.
- dspi_slave_edma_transfer_callback_t dspi_slave_edma_handle_t::callback
Completion callback.
- void * dspi_slave_edma_handle_t::userData
Callback user data.

DSPI eDMA Driver

- `edma_handle_t * dspi_slave_edma_handle_t::edmaRxRegToRxDataHandle`
edma_handle_t handle point used for RxReg to RxData buff
- `edma_handle_t * dspi_slave_edma_handle_t::edmaTxDataToTxRegHandle`
edma_handle_t handle point used for TxData to TxReg

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, 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)`
Initializes the DSPI master eDMA handle.
- `status_t DSPI_MasterTransferEDMA (SPI_Type *base, dspi_master_edma_handle_t *handle, dspi_transfer_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.

14.3.2 Data Structure Documentation

14.3.2.1 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.

14.3.2.2 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

14.3.3 Macro Definition Documentation

14.3.3.1 #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

14.3.4 Typedef Documentation

14.3.4.1 **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.

14.3.4.2 **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.

14.3.5 Function Documentation

14.3.5.1 **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.

DSPI eDMA Driver

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 _dspi_master_edma_handle .
<i>callback</i>	DSPI callback.
<i>userData</i>	A callback function parameter.
<i>edmaRxRegTo-RxDataHandle</i>	<code>edmaRxRegToRxDataHandle</code> pointer to edma_handle_t .
<i>edmaTxData-To-Intermediary-Handle</i>	<code>edmaTxDataToIntermediaryHandle</code> pointer to edma_handle_t .
<i>edma-Intermediary-ToTxReg-Handle</i>	<code>edmaIntermediaryToTxRegHandle</code> pointer to edma_handle_t .

14.3.5.2 `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 _dspi_master_edma_handle structure which stores the transfer state.
<i>transfer</i>	A pointer to the dspi_transfer_t structure.

Returns

status of status_t.

14.3.5.3 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 _dspi_master_edma_handle structure which stores the transfer state.
<i>xfer</i>	A pointer to the dspi_half_duplex_transfer_t structure.

Returns

status of status_t.

14.3.5.4 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 _dspi_master_edma_handle structure which stores the transfer state.

14.3.5.5 status_t DSPI_MasterTransferGetCountEDMA (SPI_Type * *base*, dspi_master_edma_handle_t * *handle*, size_t * *count*)

This function gets the master eDMA transfer count.

DSPI eDMA Driver

Parameters

<i>base</i>	DSPI peripheral base address.
<i>handle</i>	A pointer to the _dspi_master_edma_handle structure which stores the transfer state.
<i>count</i>	A number of bytes transferred by the non-blocking transaction.

Returns

status of status_t.

**14.3.5.6 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 edmaRx-RegToRxDataHandle 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 _dspi_slave_edma_handle .
<i>callback</i>	DSPI callback.
<i>userData</i>	A callback function parameter.
<i>edmaRxRegTo-RxDataHandle</i>	edmaRxRegToRxDataHandle pointer to edma_handle_t .

<i>edmaTxDataToTxRegHandle</i>	edmaTxDataToTxRegHandle pointer to edma_handle_t .
--------------------------------	--

14.3.5.7 **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 _dspi_slave_edma_handle structure which stores the transfer state.
<i>transfer</i>	A pointer to the dspi_transfer_t structure.

Returns

status of status_t.

14.3.5.8 **void DSPI_SlaveTransferAbortEDMA (SPI_Type * *base*, dspi_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 _dspi_slave_edma_handle structure which stores the transfer state.

14.3.5.9 **status_t DSPI_SlaveTransferGetCountEDMA (SPI_Type * *base*, dspi_slave_edma_handle_t * *handle*, size_t * *count*)**

This function gets the slave eDMA transfer count.

DSPI eDMA Driver

Parameters

<i>base</i>	DSPI peripheral base address.
<i>handle</i>	A pointer to the _dspi_slave_edma_handle 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.

14.3.6 Variable Documentation

- 14.3.6.1 `uint32_t dspi_master_edma_handle_t::bitsPerFrame`
- 14.3.6.2 `volatile uint32_t dspi_master_edma_handle_t::command`
- 14.3.6.3 `volatile uint32_t dspi_master_edma_handle_t::lastCommand`
- 14.3.6.4 `uint8_t dspi_master_edma_handle_t::fifoSize`
- 14.3.6.5 `volatile bool dspi_master_edma_handle_t::isPcsActiveAfterTransfer`
- 14.3.6.6 `uint8_t dspi_master_edma_handle_t::nbytes`
- 14.3.6.7 `volatile uint8_t dspi_master_edma_handle_t::state`
- 14.3.6.8 `uint8_t* volatile dspi_master_edma_handle_t::txData`
- 14.3.6.9 `uint8_t* volatile dspi_master_edma_handle_t::rxData`
- 14.3.6.10 `volatile size_t dspi_master_edma_handle_t::remainingSendByteCount`
- 14.3.6.11 `volatile size_t dspi_master_edma_handle_t::remainingReceiveByteCount`
- 14.3.6.12 `uint32_t dspi_master_edma_handle_t::rxBuffIfNull`
- 14.3.6.13 `uint32_t dspi_master_edma_handle_t::txBuffIfNull`
- 14.3.6.14 `dspi_master_edma_transfer_callback_t dspi_master_edma_handle_t::callback`
- 14.3.6.15 `void* dspi_master_edma_handle_t::userData`
- 14.3.6.16 `uint32_t dspi_slave_edma_handle_t::bitsPerFrame`
- 14.3.6.17 `uint8_t* volatile dspi_slave_edma_handle_t::txData`
- 14.3.6.18 `uint8_t* volatile dspi_slave_edma_handle_t::rxData`
- 14.3.6.19 `volatile size_t dspi_slave_edma_handle_t::remainingSendByteCount`
- 14.3.6.20 `volatile size_t dspi_slave_edma_handle_t::remainingReceiveByteCount`
- 14.3.6.21 `uint32_t dspi_slave_edma_handle_t::rxBuffIfNull`
- 14.3.6.22 `uint32_t dspi_slave_edma_handle_t::txBuffIfNull`
- 14.3.6.23 `uint32_t dspi_slave_edma_handle_t::txLastData`
- 14.3.6.24 `uint8_t dspi_slave_edma_handle_t::nbytes`

DSPI FreeRTOS Driver

14.4.1 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.

14.4.2 Macro Definition Documentation

14.4.2.1 #define FSL_DSPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 2, 4))

14.4.3 Function Documentation

14.4.3.1 `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

<i>handle</i>	The RTOS DSPI handle, the pointer to an allocated space for RTOS context.
<i>base</i>	The pointer base address of the DSPI instance to initialize.
<i>masterConfig</i>	A configuration structure to set-up the DSPI in master mode.
<i>srcClock_Hz</i>	A frequency of the input clock of the DSPI module.

Returns

status of the operation.

14.4.3.2 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.
---------------	-----------------------

14.4.3.3 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.

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>.

14.5.1 Function groups

14.5.1.1 DSPI CMSIS GetVersion Operation

This function group will return the DSPI CMSIS Driver version to user.

14.5.1.2 DSPI CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

14.5.1.3 DSPI CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialize 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.

14.5.1.4 DSPI CMSIS Transfer Operation

This function group controls the transfer, master send/receive data, and slave send/receive data.

14.5.1.5 DSPI CMSIS Status Operation

This function group gets the DSPI transfer status.

14.5.1.6 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.

14.5.2 Typical use case

14.5.2.1 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 uninitialized */
Driver_SPI0.Uninitialize();

```

14.5.2.2 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 uninitialized */
Driver_SPI1.Uninitialize();

```


Chapter 15

eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

Overview

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

Typical use case

15.2.1 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, 3, 2))

Typical use case

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.

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](#) type, 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_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.

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](#) type, [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.*

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_PrepareTransferConfig](#) ([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_PrepareTransfer](#) ([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](#) type)

Data Structure Documentation

- 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.

Data Structure Documentation

15.3.1 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.

15.3.1.0.0.26 Field Documentation

15.3.1.0.0.26.1 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.

15.3.1.0.0.26.2 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.

15.3.1.0.0.26.3 bool edma_config_t::enableDebugMode

When in debug mode, the eDMA stalls the start of a new channel. Executing channels are allowed to complete.

15.3.2 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.

15.3.2.0.0.27 Field Documentation

15.3.2.0.0.27.1 uint32_t edma_transfer_config_t::srcAddr

15.3.2.0.0.27.2 uint32_t edma_transfer_config_t::destAddr

15.3.2.0.0.27.3 edma_transfer_size_t edma_transfer_config_t::srcTransferSize

15.3.2.0.0.27.4 edma_transfer_size_t edma_transfer_config_t::destTransferSize

15.3.2.0.0.27.5 int16_t edma_transfer_config_t::srcOffset

15.3.2.0.0.27.6 int16_t edma_transfer_config_t::destOffset

15.3.2.0.0.27.7 uint32_t edma_transfer_config_t::majorLoopCounts

15.3.3 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](#)

Data Structure Documentation

- *If true: a channel can suspend other channel with low priority.*
uint8_t [channelPriority](#)
Channel priority.

15.3.4 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.

15.3.4.0.0.28 Field Documentation

15.3.4.0.0.28.1 bool edma_minor_offset_config_t::enableSrcMinorOffset

15.3.4.0.0.28.2 bool edma_minor_offset_config_t::enableDestMinorOffset

15.3.4.0.0.28.3 uint32_t edma_minor_offset_config_t::minorOffset

15.3.5 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.

15.3.5.0.0.29 Field Documentation

15.3.5.0.0.29.1 `__IO uint16_t edma_tcd_t::CITER`

15.3.5.0.0.29.2 `__IO uint16_t edma_tcd_t::BITER`

15.3.6 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.

Typedef Documentation

15.3.6.0.0.30 Field Documentation

15.3.6.0.0.30.1 `edma_callback edma_handle_t::callback`

15.3.6.0.0.30.2 `void* edma_handle_t::userData`

15.3.6.0.0.30.3 `DMA_Type* edma_handle_t::base`

15.3.6.0.0.30.4 `edma_tcd_t* edma_handle_t::tcdPool`

15.3.6.0.0.30.5 `uint8_t edma_handle_t::channel`

15.3.6.0.0.30.6 `volatile int8_t edma_handle_t::header`

Should point to the next TCD to be loaded into the eDMA engine.

15.3.6.0.0.30.7 `volatile int8_t edma_handle_t::tail`

Should point to the next TCD to be stored into the memory pool.

15.3.6.0.0.30.8 `volatile int8_t edma_handle_t::tcdUsed`

Should reflect the number of TCDs can be used/loaded in the memory.

15.3.6.0.0.30.9 `volatile int8_t edma_handle_t::tcdSize`

15.3.6.0.0.30.10 `uint8_t edma_handle_t::flags`

Macro Definition Documentation

15.4.1 `#define FSL_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 3, 2))`

Version 2.3.2.

Typedef Documentation

15.5.1 `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 EDMA 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.

Enumeration Type Documentation

15.6.1 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.

15.6.2 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.

Enumeration Type Documentation

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.

15.6.3 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.

15.6.4 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.

15.6.5 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

15.6.6 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.

15.6.7 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.

15.6.8 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.

15.6.9 anonymous enum

Enumerator

kStatus_EDMA_QueueFull TCD queue is full.

kStatus_EDMA_Busy Channel is busy and can't handle the transfer request.

Function Documentation

Function Documentation

15.7.1 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.

15.7.2 void EDMA_Deinit (DMA_Type * *base*)

This function gates the eDMA clock.

Parameters

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

15.7.3 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.

15.7.4 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;  
*
```

Function Documentation

Parameters

<i>config</i>	A pointer to the eDMA configuration structure.
---------------	--

15.7.5 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.

15.7.6 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, &stcd);
*
```

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.

15.7.7 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.

15.7.8 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.

15.7.9 void EDMA_SetChannelLink (DMA_Type * *base*, uint32_t *channel*, edma_channel_link_type_t *type*, 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.

Function Documentation

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>type</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.

15.7.10 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

15.7.11 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.

15.7.12 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).

15.7.13 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.

15.7.14 void EDMA_DisableChannelInterrupts (DMA_Type * *base*, uint32_t *channel*, uint32_t *mask*)

Parameters

Function Documentation

<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 <code>edma_interrupt_enable_t</code> type.

15.7.15 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.

15.7.16 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 STCD 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.

15.7.17 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.

15.7.18 void EDMA_TcdSetChannelLink (edma_tcd_t * *tcd*, edma_channel_link_type_t *type*, 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>type</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.

Function Documentation

15.7.19 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

15.7.20 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.

15.7.21 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).

15.7.22 void EDMA_TcdEnableInterrupts (edma_tcd_t * *tcd*, uint32_t *mask*)

Function Documentation

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 <code>edma_interrupt_enable_t</code> type.

15.7.23 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 <code>edma_interrupt_enable_t</code> type.

15.7.24 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.

15.7.25 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.

15.7.26 `static void EDMA_TriggerChannelStart (DMA_Type * base, uint32_t channel) [inline], [static]`

This function starts a minor loop transfer.

Function Documentation

Parameters

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

15.7.27 **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 * NBYTES(initially configured)

15.7.28 **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.

15.7.29 `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.

15.7.30 `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.

15.7.31 `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.

Function Documentation

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.

15.7.32 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_Submit-Transfer.

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.

15.7.33 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.

15.7.34 void EDMA_PrepareTransferConfig (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.

Function Documentation

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>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>bytesEach-Request</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).

15.7.35 void EDMA_PrepareTransfer (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 *type*)

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>bytesEach-Request</i>	eDMA transfer bytes per channel request.
<i>transferBytes</i>	eDMA transfer bytes to be transferred.
<i>type</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).

15.7.36 **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.

15.7.37 **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.
---------------	----------------------

15.7.38 **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.

Function Documentation

Parameters

<i>handle</i>	eDMA handle pointer.
---------------	----------------------

15.7.39 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.
---------------	---------------------

15.7.40 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.

15.7.41 static uint32_t EDMA_GetNextTCDAddress (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.

15.7.42 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 16

ENET: Ethernet MAC Driver

Overview

The MCUXpresso SDK provides a peripheral driver for the 10/100 Mbps Ethernet MAC (ENET) module of MCUXpresso SDK devices.

ENET: Ethernet MAC Driver {EthernetMACDriver}

Operations of Ethernet MAC Driver

16.2.1 MII interface Operation

The MII interface is the interface connected with MAC and PHY. the Serial management interface - MII management interface should be set before any access to the external PHY chip register. Call [ENET_SetSMI\(\)](#) to initialize the MII management interface. Use [ENET_StartSMIRead\(\)](#), [ENET_StartSMIWrite\(\)](#), and [ENET_ReadSMIData\(\)](#) to read/write to PHY registers. This function group sets up the MII and serial management SMI interface, gets data from the SMI interface, and starts the SMI read and write command. Use [ENET_SetMII\(\)](#) to configure the MII before successfully getting data from the external PHY.

16.2.2 MAC address filter

This group sets/gets the ENET mac address and the multicast group address filter. [ENET_AddMulticastGroup\(\)](#) should be called to add the ENET MAC to the multicast group. The IEEE 1588 feature requires receiving the PTP message.

16.2.3 Other Basic control Operations

This group has the receive active API [ENET_ActiveRead\(\)](#) for single and multiple rings. The [ENET_AVBConfigure\(\)](#) is provided to configure the AVB features to support the AVB frames transmission. Note that due to the AVB frames transmission scheme being a credit-based TX scheme, it is only supported with the Enhanced buffer descriptors. Because of this, the AVB configuration should only be done with the Enhanced buffer descriptor. When the AVB feature is required, make sure the "ENET_ENHANCEDBUFFERDESCRIPTOR_MODE" is defined before using this feature.

16.2.4 Transactional Operation

For ENET receive, the [ENET_GetRxFrameSize\(\)](#) function needs to be called to get the received data size. Then, call the [ENET_ReadFrame\(\)](#) function to get the received data. If the received error occurs, call the

Typical use case

[ENET_GetRxErrBeforeReadFrame\(\)](#) function after [ENET_GetRxFrameSize\(\)](#) and before [ENET_ReadFrame\(\)](#) functions to get the detailed error information.

For ENET transmit, call the [ENET_SendFrame\(\)](#) function to send the data out. The transmit data error information is only accessible for the IEEE 1588 enhanced buffer descriptor mode. When the `ENET_ENHANCEDBUFFERDESCRIPTOR_MODE` is defined, the [ENET_GetTxErrAfterSendFrame\(\)](#) can be used to get the detail transmit error information. The transmit error information can only be updated by uDMA after the data is transmitted. The [ENET_GetTxErrAfterSendFrame\(\)](#) function is recommended to be called on the transmit interrupt handler.

If send/read frame with zero-copy mechanism is needed, there're special APIs like [ENET_GetRxBuffer\(\)](#), [ENET_ReleaseRxBuffer\(\)](#), [ENET_SendFrameZeroCopy\(\)](#) and [ENET_SetTxBuffer\(\)](#). The send frame zero-copy APIs can't be used mixed with [ENET_SendFrame\(\)](#) for the same ENET peripheral, same as read frame zero-copy APIs.

16.2.5 PTP IEEE 1588 Feature Operation

This function group configures the PTP IEEE 1588 feature, starts/stops/gets/sets/adjusts the PTP IEEE 1588 timer, gets the receive/transmit frame timestamp, and PTP IEEE 1588 timer channel feature setting.

The [ENET_Ptp1588Configure\(\)](#) function needs to be called when the `ENET_ENHANCEDBUFFERDESCRIPTOR_MODE` is defined and the IEEE 1588 feature is required.

Typical use case

16.3.1 ENET Initialization, receive, and transmit operations

For the `ENET_ENHANCEDBUFFERDESCRIPTOR_MODE` undefined use case, use the legacy type buffer descriptor transmit/receive the frame as follows. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/enet` For the `ENET_ENHANCEDBUFFERDESCRIPTOR_MODE` defined use case, add the PTP IEEE 1588 configuration to enable the PTP IEEE 1588 feature. The initialization occurs as follows. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/enet`

Modules

- [ENET CMSIS Driver](#)

Data Structures

- struct [enet_rx_bd_struct_t](#)
Defines the receive buffer descriptor structure for the little endian system. [More...](#)
- struct [enet_tx_bd_struct_t](#)
Defines the enhanced transmit buffer descriptor structure for the little endian system. [More...](#)
- struct [enet_data_error_stats_t](#)
Defines the ENET data error statistics structure. [More...](#)
- struct [enet_frame_info_t](#)

- Defines the frame info structure. [More...](#)
- struct [enet_tx_dirty_ring_t](#)
Defines the ENET transmit dirty addresses ring/queue structure. [More...](#)
- struct [enet_buffer_config_t](#)
Defines the receive buffer descriptor configuration structure. [More...](#)
- struct [enet_config_t](#)
Defines the basic configuration structure for the ENET device. [More...](#)
- struct [enet_tx_bd_ring_t](#)
Defines the ENET transmit buffer descriptor ring/queue structure. [More...](#)
- struct [enet_rx_bd_ring_t](#)
Defines the ENET receive buffer descriptor ring/queue structure. [More...](#)
- struct [enet_handle_t](#)
Defines the ENET handler structure. [More...](#)

Macros

- #define [ENET_BUFFDESCRIPTOR_RX_ERR_MASK](#)
Defines the receive error status flag mask.

Typedefs

- typedef void(* [enet_callback_t](#))(ENET_Type *base, enet_handle_t *handle, [enet_event_t](#) event, [enet_frame_info_t](#) *frameInfo, void *userData)
ENET callback function.
- typedef void(* [enet_isr_t](#))(ENET_Type *base, enet_handle_t *handle)
Define interrupt IRQ handler.

Enumerations

- enum {
[kStatus_ENET_RxFrameError](#) = MAKE_STATUS(kStatusGroup_ENET, 0U),
[kStatus_ENET_RxFrameFail](#) = MAKE_STATUS(kStatusGroup_ENET, 1U),
[kStatus_ENET_RxFrameEmpty](#) = MAKE_STATUS(kStatusGroup_ENET, 2U),
[kStatus_ENET_TxFrameOverLen](#) = MAKE_STATUS(kStatusGroup_ENET, 3U),
[kStatus_ENET_TxFrameBusy](#) = MAKE_STATUS(kStatusGroup_ENET, 4U),
[kStatus_ENET_TxFrameFail](#) = MAKE_STATUS(kStatusGroup_ENET, 5U) }
Defines the status return codes for transaction.
- enum [enet_mii_mode_t](#) {
[kENET_MiiMode](#) = 0U,
[kENET_RmiiMode](#) = 1U }
Defines the MII/RMII/RGMII mode for data interface between the MAC and the PHY.
- enum [enet_mii_speed_t](#) {
[kENET_MiiSpeed10M](#) = 0U,
[kENET_MiiSpeed100M](#) = 1U }
Defines the 10/100/1000 Mbps speed for the MII data interface.
- enum [enet_mii_duplex_t](#) {
[kENET_MiiHalfDuplex](#) = 0U,
[kENET_MiiFullDuplex](#) }
Defines the half or full duplex for the MII data interface.

Typical use case

- enum `enet_mii_write_t` {
 `kENET_MiiWriteNoCompliant` = 0U,
 `kENET_MiiWriteValidFrame` }
 Define the MII opcode for normal MDIO_CLAUSES_22 Frame.
- enum `enet_mii_read_t` {
 `kENET_MiiReadValidFrame` = 2U,
 `kENET_MiiReadNoCompliant` = 3U }
 Defines the read operation for the MII management frame.
- enum `enet_special_control_flag_t` {
 `kENET_ControlFlowControlEnable` = 0x0001U,
 `kENET_ControlRxPayloadCheckEnable` = 0x0002U,
 `kENET_ControlRxPadRemoveEnable` = 0x0004U,
 `kENET_ControlRxBroadCastRejectEnable` = 0x0008U,
 `kENET_ControlMacAddrInsert` = 0x0010U,
 `kENET_ControlStoreAndFwdDisable` = 0x0020U,
 `kENET_ControlSMIPreambleDisable` = 0x0040U,
 `kENET_ControlPromiscuousEnable` = 0x0080U,
 `kENET_ControlMIILoopEnable` = 0x0100U,
 `kENET_ControlVLANTagEnable` = 0x0200U }
 Defines a special configuration for ENET MAC controller.
- enum `enet_interrupt_enable_t` {
 `kENET_BabrInterrupt` = ENET_EIR_BABR_MASK,
 `kENET_BabtInterrupt` = ENET_EIR_BABT_MASK,
 `kENET_GraceStopInterrupt` = ENET_EIR_GRA_MASK,
 `kENET_TxFrameInterrupt` = ENET_EIR_TXF_MASK,
 `kENET_TxBufferInterrupt` = ENET_EIR_TXB_MASK,
 `kENET_RxFrameInterrupt` = ENET_EIR_RXF_MASK,
 `kENET_RxBufferInterrupt` = ENET_EIR_RXB_MASK,
 `kENET_MiiInterrupt` = ENET_EIR_MII_MASK,
 `kENET_EBusERInterrupt` = ENET_EIR_EBERR_MASK,
 `kENET_LateCollisionInterrupt` = ENET_EIR_LC_MASK,
 `kENET_RetryLimitInterrupt` = ENET_EIR_RL_MASK,
 `kENET_UnderrunInterrupt` = ENET_EIR_UN_MASK,
 `kENET_PayloadRxInterrupt` = ENET_EIR_PLR_MASK,
 `kENET_WakeupInterrupt` = ENET_EIR_WAKEUP_MASK,
 `kENET_TsAvailInterrupt` = ENET_EIR_TS_AVAIL_MASK,
 `kENET_TsTimerInterrupt` = ENET_EIR_TS_TIMER_MASK }
 List of interrupts supported by the peripheral.
- enum `enet_event_t` {
 `kENET_RxEvent`,
 `kENET_TxEvent`,
 `kENET_ErrEvent`,
 `kENET_WakeUpEvent`,
 `kENET_TimeStampEvent`,
 `kENET_TimeStampAvailEvent` }
 Defines the common interrupt event for callback use.

- enum `enet_tx_accelerator_t` {
`kENET_TxAccelIsShift16Enabled` = `ENET_TACC_SHIFT16_MASK`,
`kENET_TxAccelIpCheckEnabled` = `ENET_TACC_IPCHK_MASK`,
`kENET_TxAccelProtoCheckEnabled` = `ENET_TACC_PROCHK_MASK` }
Defines the transmit accelerator configuration.
- enum `enet_rx_accelerator_t` {
`kENET_RxAccelPadRemoveEnabled` = `ENET_RACC_PADREM_MASK`,
`kENET_RxAccelIpCheckEnabled` = `ENET_RACC_IPDIS_MASK`,
`kENET_RxAccelProtoCheckEnabled` = `ENET_RACC_PRODIS_MASK`,
`kENET_RxAccelMacCheckEnabled` = `ENET_RACC_LINEDIS_MASK`,
`kENET_RxAccelIsShift16Enabled` = `ENET_RACC_SHIFT16_MASK` }
Defines the receive accelerator configuration.

Functions

- `uint32_t ENET_GetInstance` (`ENET_Type *base`)
Get the ENET instance from peripheral base address.

Driver version

- `#define FSL_ENET_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 0)`)
Defines the driver version.

ENET DESCRIPTOR QUEUE

- `#define FSL_FEATURE_ENET_QUEUE` 1 /* Singal queue for previous IP. */
Defines the queue number.

Control and status region bit masks of the receive buffer descriptor.

- `#define ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK` 0x8000U
Empty bit mask.
- `#define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK` 0x4000U
Software owner one mask.
- `#define ENET_BUFFDESCRIPTOR_RX_WRAP_MASK` 0x2000U
Next buffer descriptor is the start address.
- `#define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask` 0x1000U
Software owner two mask.
- `#define ENET_BUFFDESCRIPTOR_RX_LAST_MASK` 0x0800U
Last BD of the frame mask.
- `#define ENET_BUFFDESCRIPTOR_RX_MISS_MASK` 0x0100U
Received because of the promiscuous mode.
- `#define ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK` 0x0080U
Broadcast packet mask.
- `#define ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK` 0x0040U
Multicast packet mask.
- `#define ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK` 0x0020U
Length violation mask.
- `#define ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK` 0x0010U

Typical use case

- *Non-octet aligned frame mask.*
• #define [ENET_BUFFDESCRIPTOR_RX_CRC_MASK](#) 0x0004U
- *CRC error mask.*
• #define [ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK](#) 0x0002U
- *FIFO overrun mask.*
• #define [ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK](#) 0x0001U
- *Frame is truncated mask.*

Control and status bit masks of the transmit buffer descriptor.

- #define [ENET_BUFFDESCRIPTOR_TX_READY_MASK](#) 0x8000U
Ready bit mask.
- #define [ENET_BUFFDESCRIPTOR_TX_SOFTOWNER1_MASK](#) 0x4000U
Software owner one mask.
- #define [ENET_BUFFDESCRIPTOR_TX_WRAP_MASK](#) 0x2000U
Wrap buffer descriptor mask.
- #define [ENET_BUFFDESCRIPTOR_TX_SOFTOWNER2_MASK](#) 0x1000U
Software owner two mask.
- #define [ENET_BUFFDESCRIPTOR_TX_LAST_MASK](#) 0x0800U
Last BD of the frame mask.
- #define [ENET_BUFFDESCRIPTOR_TX_TRANSMITCRC_MASK](#) 0x0400U
Transmit CRC mask.

Defines some Ethernet parameters.

- #define [ENET_FRAME_MAX_FRAMELEN](#) 1518U
Default maximum Ethernet frame size.
- #define [ENET_FIFO_MIN_RX_FULL](#) 5U
ENET minimum receive FIFO full.
- #define [ENET_RX_MIN_BUFFERSIZE](#) 256U
ENET minimum buffer size.
- #define [ENET_PHY_MAXADDRESS](#) (ENET_MMFR_PA_MASK >> ENET_MMFR_PA_SHIFT)
Maximum PHY address.
- #define [ENET_TX_INTERRUPT](#) ((uint32_t)kENET_TxFrameInterrupt | (uint32_t)kENET_Tx-BufferInterrupt)
Enet Tx interrupt flag.
- #define [ENET_RX_INTERRUPT](#) ((uint32_t)kENET_RxFrameInterrupt | (uint32_t)kENET_Rx-BufferInterrupt)
Enet Rx interrupt flag.
- #define [ENET_TS_INTERRUPT](#) ((uint32_t)kENET_TsTimerInterrupt | (uint32_t)kENET_Ts-AvailInterrupt)
Enet timestamp interrupt flag.
- #define [ENET_ERR_INTERRUPT](#)
Enet error interrupt flag.

Defines Tx operation flags.

- #define [ENET_TX_LAST_BD_FLAG](#) 0x01U
Tx set last buffer descriptor flag.

- #define `ENET_TX_TIMESTAMP_FLAG` 0x02U
Tx timestamp flag.

Initialization and De-initialization

- void `ENET_GetDefaultConfig` (`enet_config_t` *config)
Gets the ENET default configuration structure.
- void `ENET_Up` (`ENET_Type` *base, `enet_handle_t` *handle, const `enet_config_t` *config, const `enet_buffer_config_t` *bufferConfig, `uint8_t` *macAddr, `uint32_t` srcClock_Hz)
Initializes the ENET module.
- void `ENET_Init` (`ENET_Type` *base, `enet_handle_t` *handle, const `enet_config_t` *config, const `enet_buffer_config_t` *bufferConfig, `uint8_t` *macAddr, `uint32_t` srcClock_Hz)
Initializes the ENET module.
- void `ENET_Down` (`ENET_Type` *base)
Stops the ENET module.
- void `ENET_Deinit` (`ENET_Type` *base)
Deinitializes the ENET module.
- static void `ENET_Reset` (`ENET_Type` *base)
Resets the ENET module.

MII interface operation

- void `ENET_SetMII` (`ENET_Type` *base, `enet_mii_speed_t` speed, `enet_mii_duplex_t` duplex)
Sets the ENET MII speed and duplex.
- void `ENET_SetSMI` (`ENET_Type` *base, `uint32_t` srcClock_Hz, bool isPreambleDisabled)
Sets the ENET SMI(serial management interface)- MII management interface.
- static bool `ENET_GetSMI` (`ENET_Type` *base)
Gets the ENET SMI- MII management interface configuration.
- static `uint32_t` `ENET_ReadSMIData` (`ENET_Type` *base)
Reads data from the PHY register through an SMI interface.
- void `ENET_StartSMIRead` (`ENET_Type` *base, `uint32_t` phyAddr, `uint32_t` phyReg, `enet_mii_read_t` operation)
Starts an SMI (Serial Management Interface) read command.
- void `ENET_StartSMIWrite` (`ENET_Type` *base, `uint32_t` phyAddr, `uint32_t` phyReg, `enet_mii_write_t` operation, `uint32_t` data)
Starts an SMI write command.

MAC Address Filter

- void `ENET_SetMacAddr` (`ENET_Type` *base, `uint8_t` *macAddr)
Sets the ENET module Mac address.
- void `ENET_GetMacAddr` (`ENET_Type` *base, `uint8_t` *macAddr)
Gets the ENET module Mac address.
- void `ENET_AddMulticastGroup` (`ENET_Type` *base, `uint8_t` *address)
Adds the ENET device to a multicast group.
- void `ENET_LeaveMulticastGroup` (`ENET_Type` *base, `uint8_t` *address)
Moves the ENET device from a multicast group.

Other basic operation

- static void `ENET_ActiveRead` (`ENET_Type` *base)

Typical use case

- *Activates ENET read or receive.*
static void [ENET_EnableSleepMode](#) (ENET_Type *base, bool enable)
Enables/disables the MAC to enter sleep mode.
- static void [ENET_GetAccelFunction](#) (ENET_Type *base, uint32_t *txAccelOption, uint32_t *rxAccelOption)
Gets ENET transmit and receive accelerator functions from MAC controller.

Interrupts.

- static void [ENET_EnableInterrupts](#) (ENET_Type *base, uint32_t mask)
Enables the ENET interrupt.
- static void [ENET_DisableInterrupts](#) (ENET_Type *base, uint32_t mask)
Disables the ENET interrupt.
- static uint32_t [ENET_GetInterruptStatus](#) (ENET_Type *base)
Gets the ENET interrupt status flag.
- static void [ENET_ClearInterruptStatus](#) (ENET_Type *base, uint32_t mask)
Clears the ENET interrupt events status flag.
- void [ENET_SetRxISRHandler](#) (ENET_Type *base, [enet_isr_t](#) ISRHandler)
Set the second level Rx IRQ handler.
- void [ENET_SetTxISRHandler](#) (ENET_Type *base, [enet_isr_t](#) ISRHandler)
Set the second level Tx IRQ handler.
- void [ENET_SetErrISRHandler](#) (ENET_Type *base, [enet_isr_t](#) ISRHandler)
Set the second level Err IRQ handler.

Transactional operation

- void [ENET_SetCallback](#) (enet_handle_t *handle, [enet_callback_t](#) callback, void *userData)
Sets the callback function.
- void [ENET_GetRxErrBeforeReadFrame](#) (enet_handle_t *handle, [enet_data_error_stats_t](#) *eError-Static, uint8_t ringId)
Gets the error statistics of a received frame for ENET specified ring.
- [status_t](#) [ENET_GetRxFrameSize](#) (enet_handle_t *handle, uint32_t *length, uint8_t ringId)
Gets the size of the read frame for specified ring.
- [status_t](#) [ENET_ReadFrame](#) (ENET_Type *base, enet_handle_t *handle, uint8_t *data, uint32_t length, uint8_t ringId, uint32_t *ts)
Reads a frame from the ENET device.
- [status_t](#) [ENET_SendFrame](#) (ENET_Type *base, enet_handle_t *handle, const uint8_t *data, uint32_t length, uint8_t ringId, bool tsFlag, void *context)
Transmits an ENET frame for specified ring.
- [status_t](#) [ENET_SetTxReclaim](#) (enet_handle_t *handle, bool isEnabled, uint8_t ringId)
Enable or disable tx descriptors reclaim mechanism.
- [status_t](#) [ENET_GetRxBuffer](#) (ENET_Type *base, enet_handle_t *handle, void **buffer, uint32_t *length, uint8_t ringId, bool *isLastBuff, uint32_t *ts)
Get a receive buffer pointer of the ENET device for specified ring.
- void [ENET_ReleaseRxBuffer](#) (ENET_Type *base, enet_handle_t *handle, void *buffer, uint8_t ringId)
Release receive buffer descriptor to DMA.
- [status_t](#) [ENET_SendFrameZeroCopy](#) (ENET_Type *base, enet_handle_t *handle, const uint8_t *data, uint32_t length, uint8_t ringId, bool tsFlag, void *context)
Transmits an ENET frame for specified ring with zero-copy.

- [status_t ENET_SetTxBuffer](#) (ENET_Type *base, enet_handle_t *handle, const uint8_t *data, uint32_t length, uint8_t ringId, uint8_t txFlag, void *context)
Set up ENET Tx buffer descriptor, preparing for one frame stores in scattered buffer.
- void [ENET_TransmitIRQHandler](#) (ENET_Type *base, enet_handle_t *handle)
The transmit IRQ handler.
- void [ENET_ReceiveIRQHandler](#) (ENET_Type *base, enet_handle_t *handle)
The receive IRQ handler.
- void [ENET_ErrorIRQHandler](#) (ENET_Type *base, enet_handle_t *handle)
Some special IRQ handler including the error, mii, wakeup irq handler.
- void [ENET_CommonFrame0IRQHandler](#) (ENET_Type *base)
the common IRQ handler for the tx/rx/error etc irq handler.

Data Structure Documentation

16.4.1 struct enet_rx_bd_struct_t

Data Fields

- uint16_t [length](#)
Buffer descriptor data length.
- uint16_t [control](#)
Buffer descriptor control and status.
- uint8_t * [buffer](#)
Data buffer pointer.

16.4.1.0.0.31 Field Documentation

16.4.1.0.0.31.1 uint16_t enet_rx_bd_struct_t::length

16.4.1.0.0.31.2 uint16_t enet_rx_bd_struct_t::control

16.4.1.0.0.31.3 uint8_t* enet_rx_bd_struct_t::buffer

16.4.2 struct enet_tx_bd_struct_t

Data Fields

- uint16_t [length](#)
Buffer descriptor data length.
- uint16_t [control](#)
Buffer descriptor control and status.
- uint8_t * [buffer](#)
Data buffer pointer.

Data Structure Documentation

16.4.2.0.0.32 Field Documentation

16.4.2.0.0.32.1 `uint16_t enet_tx_bd_struct_t::length`

16.4.2.0.0.32.2 `uint16_t enet_tx_bd_struct_t::control`

16.4.2.0.0.32.3 `uint8_t* enet_tx_bd_struct_t::buffer`

16.4.3 `struct enet_data_error_stats_t`

Data Fields

- `uint32_t statsRxLenGreaterErr`
Receive length greater than RCR[MAX_FL].
- `uint32_t statsRxAlignErr`
Receive non-octet alignment/.
- `uint32_t statsRxFcsErr`
Receive CRC error.
- `uint32_t statsRxOverRunErr`
Receive over run.
- `uint32_t statsRxTruncateErr`
Receive truncate.

16.4.3.0.0.33 Field Documentation

16.4.3.0.0.33.1 `uint32_t enet_data_error_stats_t::statsRxLenGreaterErr`

16.4.3.0.0.33.2 `uint32_t enet_data_error_stats_t::statsRxFcsErr`

16.4.3.0.0.33.3 `uint32_t enet_data_error_stats_t::statsRxOverRunErr`

16.4.3.0.0.33.4 `uint32_t enet_data_error_stats_t::statsRxTruncateErr`

16.4.4 `struct enet_frame_info_t`

Data Fields

- `void * context`
User specified data.

16.4.5 `struct enet_tx_dirty_ring_t`

Data Fields

- `enet_frame_info_t * txDirtyBase`
Dirty buffer descriptor base address pointer.
- `uint16_t txGenIdx`

- *tx generate index.*
uint16_t [txConsumIdx](#)
- *tx consume index.*
uint16_t [txRingLen](#)
- *tx ring length.*
bool [isFull](#)
- *tx ring is full flag.*

16.4.5.0.0.34 Field Documentation

16.4.5.0.0.34.1 [enet_frame_info_t* enet_tx_dirty_ring_t::txDirtyBase](#)

16.4.5.0.0.34.2 [uint16_t enet_tx_dirty_ring_t::txGenIdx](#)

16.4.5.0.0.34.3 [uint16_t enet_tx_dirty_ring_t::txConsumIdx](#)

16.4.5.0.0.34.4 [uint16_t enet_tx_dirty_ring_t::txRingLen](#)

16.4.5.0.0.34.5 [bool enet_tx_dirty_ring_t::isFull](#)

16.4.6 struct [enet_buffer_config_t](#)

Note that for the internal DMA requirements, the buffers have a corresponding alignment requirements.

1. The aligned receive and transmit buffer size must be evenly divisible by ENET_BUFF_ALIGNMENT. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET_BUFF_ALIGNMENT" and the cache line size.
2. The aligned transmit and receive buffer descriptor start address must be at least 64 bit aligned. However, it's recommended to be evenly divisible by ENET_BUFF_ALIGNMENT. buffer descriptors should be put in non-cacheable region when cache is enabled.
3. The aligned transmit and receive data buffer start address must be evenly divisible by ENET_BUFF_ALIGNMENT. Receive buffers should be continuous with the total size equal to "rxBdNumber * rxBuffSizeAlign". Transmit buffers should be continuous with the total size equal to "txBdNumber * txBuffSizeAlign". when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET_BUFF_ALIGNMENT" and the cache line size.

Data Fields

- uint16_t [rxBdNumber](#)
Receive buffer descriptor number.
- uint16_t [txBdNumber](#)
Transmit buffer descriptor number.
- uint16_t [rxBuffSizeAlign](#)
Aligned receive data buffer size.
- uint16_t [txBuffSizeAlign](#)
Aligned transmit data buffer size.
- volatile [enet_rx_bd_struct_t * rxBdStartAddrAlign](#)

Data Structure Documentation

- Aligned receive buffer descriptor start address: should be non-cacheable.*
 - volatile [enet_tx_bd_struct_t](#) * [txBdStartAddrAlign](#)
- Aligned transmit buffer descriptor start address: should be non-cacheable.*
 - uint8_t * [rxBufferAlign](#)
- Receive data buffer start address.*
 - uint8_t * [txBufferAlign](#)
- Transmit data buffer start address.*
 - bool [rxMaintainEnable](#)
- Receive buffer cache maintain.*
 - bool [txMaintainEnable](#)
- Transmit buffer cache maintain.*
 - [enet_frame_info_t](#) * [txFrameInfo](#)
- Transmit frame information start address.*

16.4.6.0.0.35 Field Documentation

16.4.6.0.0.35.1 uint16_t enet_buffer_config_t::rxBdNumber

16.4.6.0.0.35.2 uint16_t enet_buffer_config_t::txBdNumber

16.4.6.0.0.35.3 uint16_t enet_buffer_config_t::rxBuffSizeAlign

16.4.6.0.0.35.4 uint16_t enet_buffer_config_t::txBuffSizeAlign

16.4.6.0.0.35.5 volatile enet_rx_bd_struct_t* enet_buffer_config_t::rxBdStartAddrAlign

16.4.6.0.0.35.6 volatile enet_tx_bd_struct_t* enet_buffer_config_t::txBdStartAddrAlign

16.4.6.0.0.35.7 uint8_t* enet_buffer_config_t::rxBufferAlign

16.4.6.0.0.35.8 uint8_t* enet_buffer_config_t::txBufferAlign

16.4.6.0.0.35.9 bool enet_buffer_config_t::rxMaintainEnable

16.4.6.0.0.35.10 bool enet_buffer_config_t::txMaintainEnable

16.4.6.0.0.35.11 enet_frame_info_t* enet_buffer_config_t::txFrameInfo

16.4.7 struct enet_config_t

Note:

- macSpecialConfig is used for a special control configuration, A logical OR of "enet_special_control_flag_t". For a special configuration for MAC, set this parameter to 0.
- txWatermark is used for a cut-through operation. It is in steps of 64 bytes: 0/1 - 64 bytes written to TX FIFO before transmission of a frame begins. 2 - 128 bytes written to TX FIFO 3 - 192 bytes written to TX FIFO The maximum of txWatermark is 0x2F - 4032 bytes written to TX FIFO txWatermark allows minimizing the transmit latency to set the txWatermark to 0 or 1 or for larger bus access latency 3 or larger due to contention for the system bus.
- rxFifoFullThreshold is similar to the txWatermark for cut-through operation in RX. It is in 64-bit

words. The minimum is ENET_FIFO_MIN_RX_FULL and the maximum is 0xFF. If the end of the frame is stored in FIFO and the frame size is smaller than the txWatermark, the frame is still transmitted. The rule is the same for rxFifoFullThreshold in the receive direction.

4. When "kENET_ControlFlowControlEnable" is set in the macSpecialConfig, ensure that the pauseDuration, rxFifoEmptyThreshold, and rxFifoStatEmptyThreshold are set for flow control enabled case.
5. When "kENET_ControlStoreAndFwdDisabled" is set in the macSpecialConfig, ensure that the rxFifoFullThreshold and txFifoWatermark are set for store and forward disable.
6. The rxAccelerConfig and txAccelerConfig default setting with 0 - accelerator are disabled. The "enet_tx_accelerator_t" and "enet_rx_accelerator_t" are recommended to be used to enable the transmit and receive accelerator. After the accelerators are enabled, the store and forward feature should be enabled. As a result, kENET_ControlStoreAndFwdDisabled should not be set.
7. The intCoalesceCfg can be used in the rx or tx enabled cases to decrease the CPU loading.

Data Fields

- uint32_t [macSpecialConfig](#)
Mac special configuration.
- uint32_t [interrupt](#)
Mac interrupt source.
- uint16_t [rxMaxFrameLen](#)
Receive maximum frame length.
- [enet_mii_mode_t](#) [miiMode](#)
MII mode.
- [enet_mii_speed_t](#) [miiSpeed](#)
MII Speed.
- [enet_mii_duplex_t](#) [miiDuplex](#)
MII duplex.
- uint8_t [rxAccelerConfig](#)
Receive accelerator, A logical OR of "enet_rx_accelerator_t".
- uint8_t [txAccelerConfig](#)
Transmit accelerator, A logical OR of "enet_rx_accelerator_t".
- uint16_t [pauseDuration](#)
For flow control enabled case: Pause duration.
- uint8_t [rxFifoEmptyThreshold](#)
For flow control enabled case: when RX FIFO level reaches this value, it makes MAC generate XOFF pause frame.
- uint8_t [rxFifoStatEmptyThreshold](#)
For flow control enabled case: number of frames in the receive FIFO, independent of size, that can be accept.
- uint8_t [rxFifoFullThreshold](#)
For store and forward disable case, the data required in RX FIFO to notify the MAC receive ready status.
- uint8_t [txFifoWatermark](#)
For store and forward disable case, the data required in TX FIFO before a frame transmit start.
- uint8_t [ringNum](#)
Number of used rings.

Data Structure Documentation

16.4.7.0.0.36 Field Documentation

16.4.7.0.0.36.1 uint32_t enet_config_t::macSpecialConfig

A logical OR of "enet_special_control_flag_t".

16.4.7.0.0.36.2 uint32_t enet_config_t::interrupt

A logical OR of "enet_interrupt_enable_t".

16.4.7.0.0.36.3 uint16_t enet_config_t::rxMaxFrameLen

16.4.7.0.0.36.4 enet_mii_mode_t enet_config_t::miiMode

16.4.7.0.0.36.5 enet_mii_speed_t enet_config_t::miiSpeed

16.4.7.0.0.36.6 enet_mii_duplex_t enet_config_t::miiDuplex

16.4.7.0.0.36.7 uint8_t enet_config_t::rxAccelerConfig

16.4.7.0.0.36.8 uint8_t enet_config_t::txAccelerConfig

16.4.7.0.0.36.9 uint16_t enet_config_t::pauseDuration

16.4.7.0.0.36.10 uint8_t enet_config_t::rxFifoEmptyThreshold

16.4.7.0.0.36.11 uint8_t enet_config_t::rxFifoStatEmptyThreshold

If the limit is reached, reception continues and a pause frame is triggered.

16.4.7.0.0.36.12 uint8_t enet_config_t::rxFifoFullThreshold

16.4.7.0.0.36.13 uint8_t enet_config_t::txFifoWatermark

16.4.7.0.0.36.14 uint8_t enet_config_t::ringNum

default with 1 – single ring.

16.4.8 struct enet_tx_bd_ring_t

Data Fields

- volatile [enet_tx_bd_struct_t](#) * [txBdBase](#)
Buffer descriptor base address pointer.
- uint16_t [txGenIdx](#)
The current available transmit buffer descriptor pointer.
- uint16_t [txConsumeIdx](#)
Transmit consume index.
- volatile uint16_t [txDescUsed](#)

- *Transmit descriptor used number.*
- uint16_t [txRingLen](#)
Transmit ring length.

16.4.8.0.0.37 Field Documentation

16.4.8.0.0.37.1 volatile enet_tx_bd_struct_t* enet_tx_bd_ring_t::txBdBase

16.4.8.0.0.37.2 uint16_t enet_tx_bd_ring_t::txGenIdx

16.4.8.0.0.37.3 uint16_t enet_tx_bd_ring_t::txConsumeIdx

16.4.8.0.0.37.4 volatile uint16_t enet_tx_bd_ring_t::txDescUsed

16.4.8.0.0.37.5 uint16_t enet_tx_bd_ring_t::txRingLen

16.4.9 struct enet_rx_bd_ring_t

Data Fields

- volatile [enet_rx_bd_struct_t](#) * [rxBdBase](#)
Buffer descriptor base address pointer.
- uint16_t [rxGenIdx](#)
The current available receive buffer descriptor pointer.
- uint16_t [rxRingLen](#)
Receive ring length.

16.4.9.0.0.38 Field Documentation

16.4.9.0.0.38.1 volatile enet_rx_bd_struct_t* enet_rx_bd_ring_t::rxBdBase

16.4.9.0.0.38.2 uint16_t enet_rx_bd_ring_t::rxGenIdx

16.4.9.0.0.38.3 uint16_t enet_rx_bd_ring_t::rxRingLen

16.4.10 struct _enet_handle

Data Fields

- [enet_rx_bd_ring_t](#) [rxBdRing](#) [[FSL_FEATURE_ENET_QUEUE](#)]
Receive buffer descriptor.
- [enet_tx_bd_ring_t](#) [txBdRing](#) [[FSL_FEATURE_ENET_QUEUE](#)]
Transmit buffer descriptor.
- uint16_t [rxBuffSizeAlign](#) [[FSL_FEATURE_ENET_QUEUE](#)]
Receive buffer size alignment.
- uint16_t [txBuffSizeAlign](#) [[FSL_FEATURE_ENET_QUEUE](#)]
Transmit buffer size alignment.
- bool [rxMaintainEnable](#) [[FSL_FEATURE_ENET_QUEUE](#)]
Receive buffer cache maintain.

Macro Definition Documentation

- bool `txMaintainEnable` [FSL_FEATURE_ENET_QUEUE]
Transmit buffer cache maintain.
- uint8_t `ringNum`
Number of used rings.
- `enet_callback_t` `callback`
Callback function.
- void * `userData`
Callback function parameter.
- `enet_tx_dirty_ring_t` `txDirtyRing` [FSL_FEATURE_ENET_QUEUE]
Ring to store tx frame information.
- bool `TxReclaimEnable` [FSL_FEATURE_ENET_QUEUE]
Tx reclaim enable flag.

16.4.10.0.0.39 Field Documentation

- 16.4.10.0.0.39.1 `enet_rx_bd_ring_t` `enet_handle_t::rxBdRing` [FSL_FEATURE_ENET_QUEUE]
- 16.4.10.0.0.39.2 `enet_tx_bd_ring_t` `enet_handle_t::txBdRing` [FSL_FEATURE_ENET_QUEUE]
- 16.4.10.0.0.39.3 `uint16_t` `enet_handle_t::rxBuffSizeAlign` [FSL_FEATURE_ENET_QUEUE]
- 16.4.10.0.0.39.4 `uint16_t` `enet_handle_t::txBuffSizeAlign` [FSL_FEATURE_ENET_QUEUE]
- 16.4.10.0.0.39.5 `bool` `enet_handle_t::rxMaintainEnable` [FSL_FEATURE_ENET_QUEUE]
- 16.4.10.0.0.39.6 `bool` `enet_handle_t::txMaintainEnable` [FSL_FEATURE_ENET_QUEUE]
- 16.4.10.0.0.39.7 `uint8_t` `enet_handle_t::ringNum`
- 16.4.10.0.0.39.8 `enet_callback_t` `enet_handle_t::callback`
- 16.4.10.0.0.39.9 `void*` `enet_handle_t::userData`
- 16.4.10.0.0.39.10 `enet_tx_dirty_ring_t` `enet_handle_t::txDirtyRing` [FSL_FEATURE_ENET_QUEUE]
- 16.4.10.0.0.39.11 `bool` `enet_handle_t::TxReclaimEnable` [FSL_FEATURE_ENET_QUEUE]

Macro Definition Documentation

16.5.1 #define FSL_ENET_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

Version 2.3.0.

16.5.2 #define FSL_FEATURE_ENET_QUEUE 1 /* Singal queue for previous IP. */

16.5.3 #define ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK 0x8000U

16.5.4 #define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK 0x4000U

16.5.5 #define ENET_BUFFDESCRIPTOR_RX_WRAP_MASK 0x2000U

16.5.6 #define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask 0x1000U

16.5.7 #define ENET_BUFFDESCRIPTOR_RX_LAST_MASK 0x0800U

16.5.8 #define ENET_BUFFDESCRIPTOR_RX_MISS_MASK 0x0100U

16.5.9 #define ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK 0x0080U

16.5.10 #define ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK 0x0040U

16.5.11 #define ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK 0x0020U

16.5.12 #define ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK 0x0010U

16.5.13 #define ENET_BUFFDESCRIPTOR_RX_CRC_MASK 0x0004U

16.5.14 #define ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK 0x0002U

16.5.15 #define ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK 0x0001U

16.5.16 #define ENET_BUFFDESCRIPTOR_TX_READY_MASK 0x8000U

16.5.17 #define ENET_BUFFDESCRIPTOR_TX_SOFTOWENER1_MASK 0x4000U

16.5.18 #define ENET_BUFFDESCRIPTOR_TX_WRAP_MASK 0x2000U

16.5.19 #define ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_MASK 0x1000U

16.5.20 #define ENET_BUFFDESCRIPTOR_TX_LAST_MASK 0x0800U

16.5.21 #define ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK 0x0400U

16.5.22 #define ENET_BUFFDESCRIPTOR_RX_ERR_MASK

```
(ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK |
  ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK | \
  ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK |
  ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK |
  ENET_BUFFDESCRIPTOR_RX_CRC_MASK)
```

16.5.23 #define ENET_FRAME_MAX_FRAMELEN 1518U

16.5.24 #define ENET_FIFO_MIN_RX_FULL 5U

16.5.25 #define ENET_RX_MIN_BUFFERSIZE 256U

**16.5.26 #define ENET_PHY_MAXADDRESS (ENET_MMFR_PA_MASK >>
ENET_MMFR_PA_SHIFT)**

**16.5.27 #define ENET_TX_INTERRUPT ((uint32_t)kENET_TxFrameInterrupt |
(uint32_t)kENET_TxBufferInterrupt)**

**16.5.28 #define ENET_RX_INTERRUPT ((uint32_t)kENET_RxFrameInterrupt |
(uint32_t)kENET_RxBufferInterrupt)**

**16.5.29 #define ENET_TS_INTERRUPT ((uint32_t)kENET_TsTimerInterrupt |
(uint32_t)kENET_TsAvailInterrupt)**

16.5.30 #define ENET_ERR_INTERRUPT

Value:

```
((uint32_t)kENET_BabrInterrupt | (uint32_t)
  kENET_BabtInterrupt | (uint32_t)kENET_EBusERInterrupt | \
  (uint32_t)kENET_LateCollisionInterrupt | (uint32_t)
  kENET_RetryLimitInterrupt | \
  (uint32_t)kENET_UnderrunInterrupt | (uint32_t)
  kENET_PayloadRxInterrupt)
```

16.5.31 #define ENET_TX_LAST_BD_FLAG 0x01U

16.5.32 #define ENET_TX_TIMESTAMP_FLAG 0x02U

Enumeration Type Documentation

Typedef Documentation

16.6.1 `typedef void(* enet_callback_t)(ENET_Type *base, enet_handle_t *handle, enet_event_t event, enet_frame_info_t *frameInfo, void *userData)`

16.6.2 `typedef void(* enet_isr_t)(ENET_Type *base, enet_handle_t *handle)`

Enumeration Type Documentation

16.7.1 anonymous enum

Enumerator

kStatus_ENET_RxFrameError A frame received but data error happen.

kStatus_ENET_RxFrameFail Failed to receive a frame.

kStatus_ENET_RxFrameEmpty No frame arrive.

kStatus_ENET_TxFrameOverLen Tx frame over length.

kStatus_ENET_TxFrameBusy Tx buffer descriptors are under process.

kStatus_ENET_TxFrameFail Transmit frame fail.

16.7.2 enum enet_mii_mode_t

Enumerator

kENET_MiiMode MII mode for data interface.

kENET_RmiiMode RMII mode for data interface.

16.7.3 enum enet_mii_speed_t

Notice: "kENET_MiiSpeed1000M" only supported when mii mode is "kENET_RgmiiMode".

Enumerator

kENET_MiiSpeed10M Speed 10 Mbps.

kENET_MiiSpeed100M Speed 100 Mbps.

16.7.4 enum enet_mii_duplex_t

Enumerator

kENET_MiiHalfDuplex Half duplex mode.

kENET_MiiFullDuplex Full duplex mode.

16.7.5 enum enet_mii_write_t

Enumerator

kENET_MiiWriteNoCompliant Write frame operation, but not MII-compliant.

kENET_MiiWriteValidFrame Write frame operation for a valid MII management frame.

16.7.6 enum enet_mii_read_t

Enumerator

kENET_MiiReadValidFrame Read frame operation for a valid MII management frame.

kENET_MiiReadNoCompliant Read frame operation, but not MII-compliant.

16.7.7 enum enet_special_control_flag_t

These control flags are provided for special user requirements. Normally, these control flags are unused for ENET initialization. For special requirements, set the flags to macSpecialConfig in the [enet_config_t](#). The kENET_ControlStoreAndFwdDisable is used to disable the FIFO store and forward. FIFO store and forward means that the FIFO read/send is started when a complete frame is stored in TX/RX FIFO. If this flag is set, configure rxFifoFullThreshold and txFifoWatermark in the [enet_config_t](#).

Enumerator

kENET_ControlFlowControlEnable Enable ENET flow control: pause frame.

kENET_ControlRxPayloadCheckEnable Enable ENET receive payload length check.

kENET_ControlRxPadRemoveEnable Padding is removed from received frames.

kENET_ControlRxBroadcastRejectEnable Enable broadcast frame reject.

kENET_ControlMacAddrInsert Enable MAC address insert.

kENET_ControlStoreAndFwdDisable Enable FIFO store and forward.

kENET_ControlSMIPreambleDisable Enable SMI preamble.

kENET_ControlPromiscuousEnable Enable promiscuous mode.

kENET_ControlMIILoopEnable Enable ENET MII loop back.

kENET_ControlVLANTagEnable Enable normal VLAN (single vlan tag).

16.7.8 enum enet_interrupt_enable_t

This enumeration uses one-hot encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

Enumerator

kENET_BabrInterrupt Babbling receive error interrupt source.

Enumeration Type Documentation

kENET_BabtInterrupt Babbling transmit error interrupt source.
kENET_GraceStopInterrupt Graceful stop complete interrupt source.
kENET_TxFrameInterrupt TX FRAME interrupt source.
kENET_TxBufferInterrupt TX BUFFER interrupt source.
kENET_RxFrameInterrupt RX FRAME interrupt source.
kENET_RxBufferInterrupt RX BUFFER interrupt source.
kENET_MiiInterrupt MII interrupt source.
kENET_EBusERInterrupt Ethernet bus error interrupt source.
kENET_LateCollisionInterrupt Late collision interrupt source.
kENET_RetryLimitInterrupt Collision Retry Limit interrupt source.
kENET_UnderrunInterrupt Transmit FIFO underrun interrupt source.
kENET_PayloadRxInterrupt Payload Receive error interrupt source.
kENET_WakeupInterrupt WAKEUP interrupt source.
kENET_TsAvailInterrupt TS AVAIL interrupt source for PTP.
kENET_TsTimerInterrupt TS WRAP interrupt source for PTP.

16.7.9 enum enet_event_t

Enumerator

kENET_RxEvent Receive event.
kENET_TxEvent Transmit event.
kENET_ErrEvent Error event: BABR/BABT/EBERR/LC/RL/UN/PLR .
kENET_WakeUpEvent Wake up from sleep mode event.
kENET_TimeStampEvent Time stamp event.
kENET_TimeStampAvailEvent Time stamp available event.

16.7.10 enum enet_tx_accelerator_t

Enumerator

kENET_TxAccelIsShift16Enabled Transmit FIFO shift-16.
kENET_TxAccelIpCheckEnabled Insert IP header checksum.
kENET_TxAccelProtoCheckEnabled Insert protocol checksum.

16.7.11 enum enet_rx_accelerator_t

Enumerator

kENET_RxAccelPadRemoveEnabled Padding removal for short IP frames.
kENET_RxAccelIpCheckEnabled Discard with wrong IP header checksum.
kENET_RxAccelProtoCheckEnabled Discard with wrong protocol checksum.

kENET_RxAccelMacCheckEnabled Discard with Mac layer errors.

kENET_RxAccelShift16Enabled Receive FIFO shift-16.

Function Documentation

16.8.1 uint32_t ENET_GetInstance (ENET_Type * *base*)

Parameters

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

Returns

ENET instance.

16.8.2 void ENET_GetDefaultConfig (enet_config_t * *config*)

The purpose of this API is to get the default ENET MAC controller configure structure for [ENET_Init\(\)](#). User may use the initialized structure unchanged in [ENET_Init\(\)](#), or modify some fields of the structure before calling [ENET_Init\(\)](#). Example:

```
enet_config_t config;
ENET_GetDefaultConfig(&config);
```

Parameters

<i>config</i>	The ENET mac controller configuration structure pointer.
---------------	--

16.8.3 void ENET_Up (ENET_Type * *base*, enet_handle_t * *handle*, const enet_config_t * *config*, const enet_buffer_config_t * *bufferConfig*, uint8_t * *macAddr*, uint32_t *srcClock_Hz*)

This function initializes the module with the ENET configuration.

Note

ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET_ENHANCEDBUFFERDESCRIPTOR_MODE" and calling [ENET_Ptp1588Configure\(\)](#) to configure the 1588 feature and related buffers after calling [ENET_Up\(\)](#).

Function Documentation

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	ENET handler pointer.
<i>config</i>	ENET mac configuration structure pointer. The "enet_config_t" type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.
<i>bufferConfig</i>	ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of "ringNum" enet_buffer_config structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this bufferConfig is a buffer configure structure pointer, for multi-ring supported and used case, this bufferConfig pointer should be a buffer configure structure array pointer.
<i>macAddr</i>	ENET mac address of Ethernet device. This MAC address should be provided.
<i>srcClock_Hz</i>	The internal module clock source for MII clock.

16.8.4 void ENET_Init (ENET_Type * *base*, enet_handle_t * *handle*, const enet_config_t * *config*, const enet_buffer_config_t * *bufferConfig*, uint8_t * *macAddr*, uint32_t *srcClock_Hz*)

This function ungates the module clock and initializes it with the ENET configuration.

Note

ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET_ENHANCEDBUFFERDESCRIPTOR_MODE" and calling ENET_Ptp1588Configure() to configure the 1588 feature and related buffers after calling [ENET_Init\(\)](#).

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	ENET handler pointer.
<i>config</i>	ENET mac configuration structure pointer. The "enet_config_t" type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.
<i>bufferConfig</i>	ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of "ringNum" enet_buffer_config structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this bufferConfig is a buffer configure structure pointer, for multi-ring supported and used case, this bufferConfig pointer should be a buffer configure structure array pointer.
<i>macAddr</i>	ENET mac address of Ethernet device. This MAC address should be provided.
<i>srcClock_Hz</i>	The internal module clock source for MII clock.

16.8.5 void ENET_Down (ENET_Type * *base*)

This function disables the ENET module.

Parameters

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

16.8.6 void ENET_Deinit (ENET_Type * *base*)

This function gates the module clock, clears ENET interrupts, and disables the ENET module.

Parameters

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

16.8.7 static void ENET_Reset (ENET_Type * *base*) [inline], [static]

This function restores the ENET module to reset state. Note that this function sets all registers to reset state. As a result, the ENET module can't work after calling this function.

Function Documentation

Parameters

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

16.8.8 void ENET_SetMII (ENET_Type * *base*, enet_mii_speed_t *speed*, enet_mii_duplex_t *duplex*)

This API is provided to dynamically change the speed and duplex for MAC.

Parameters

<i>base</i>	ENET peripheral base address.
<i>speed</i>	The speed of the RMII mode.
<i>duplex</i>	The duplex of the RMII mode.

16.8.9 void ENET_SetSMI (ENET_Type * *base*, uint32_t *srcClock_Hz*, bool *isPreambleDisabled*)

Parameters

<i>base</i>	ENET peripheral base address.
<i>srcClock_Hz</i>	This is the ENET module clock frequency. Normally it's the system clock. See clock distribution.
<i>isPreamble-Disabled</i>	The preamble disable flag. <ul style="list-style-type: none">• true Enables the preamble.• false Disables the preamble.

16.8.10 static bool ENET_GetSMI (ENET_Type * *base*) [inline], [static]

This API is used to get the SMI configuration to check whether the MII management interface has been set.

Parameters

--

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

Returns

The SMI setup status true or false.

**16.8.11 static uint32_t ENET_ReadSMIData (ENET_Type * *base*) [inline],
[static]**

Parameters

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

Returns

The data read from PHY

16.8.12 void ENET_StartSMIRead (ENET_Type * *base*, uint32_t *phyAddr*, uint32_t *phyReg*, enet_mii_read_t *operation*)

Used for standard IEEE802.3 MDIO Clause 22 format.

Parameters

<i>base</i>	ENET peripheral base address.
<i>phyAddr</i>	The PHY address.
<i>phyReg</i>	The PHY register. Range from 0 ~ 31.
<i>operation</i>	The read operation.

16.8.13 void ENET_StartSMIWrite (ENET_Type * *base*, uint32_t *phyAddr*, uint32_t *phyReg*, enet_mii_write_t *operation*, uint32_t *data*)

Used for standard IEEE802.3 MDIO Clause 22 format.

Function Documentation

Parameters

<i>base</i>	ENET peripheral base address.
<i>phyAddr</i>	The PHY address.
<i>phyReg</i>	The PHY register. Range from 0 ~ 31.
<i>operation</i>	The write operation.
<i>data</i>	The data written to PHY.

16.8.14 void ENET_SetMacAddr (ENET_Type * *base*, uint8_t * *macAddr*)

Parameters

<i>base</i>	ENET peripheral base address.
<i>macAddr</i>	The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

16.8.15 void ENET_GetMacAddr (ENET_Type * *base*, uint8_t * *macAddr*)

Parameters

<i>base</i>	ENET peripheral base address.
<i>macAddr</i>	The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

16.8.16 void ENET_AddMulticastGroup (ENET_Type * *base*, uint8_t * *address*)

Parameters

<i>base</i>	ENET peripheral base address.
<i>address</i>	The six-byte multicast group address which is provided by application.

16.8.17 void ENET_LeaveMulticastGroup (ENET_Type * *base*, uint8_t * *address*)

Parameters

<i>base</i>	ENET peripheral base address.
<i>address</i>	The six-byte multicast group address which is provided by application.

16.8.18 static void ENET_ActiveRead (ENET_Type * *base*) [inline], [static]

This function is to active the enet read process.

Note

This must be called after the MAC configuration and state are ready. It must be called after the [ENET_Init\(\)](#) and [ENET_Ptp1588Configure\(\)](#). This should be called when the ENET receive required.

Parameters

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

16.8.19 static void ENET_EnableSleepMode (ENET_Type * *base*, bool *enable*) [inline], [static]

This function is used to set the MAC enter sleep mode. When entering sleep mode, the magic frame wakeup interrupt should be enabled to wake up MAC from the sleep mode and reset it to normal mode.

Parameters

<i>base</i>	ENET peripheral base address.
<i>enable</i>	True enable sleep mode, false disable sleep mode.

16.8.20 static void ENET_GetAccelFunction (ENET_Type * *base*, uint32_t * *txAccelOption*, uint32_t * *rxAccelOption*) [inline], [static]

Parameters

Function Documentation

<i>base</i>	ENET peripheral base address.
<i>txAccelOption</i>	The transmit accelerator option. The "enet_tx_accelerator_t" is recommended to be used to as the mask to get the exact the accelerator option.
<i>rxAccelOption</i>	The receive accelerator option. The "enet_rx_accelerator_t" is recommended to be used to as the mask to get the exact the accelerator option.

16.8.21 static void ENET_EnableInterrupts (ENET_Type * *base*, uint32_t *mask*) [inline], [static]

This function enables the ENET interrupt according to the provided mask. The mask is a logical OR of enumeration members. See [enet_interrupt_enable_t](#). For example, to enable the TX frame interrupt and RX frame interrupt, do the following.

```
*  ENET_EnableInterrupts(ENET, kENET_TxFrameInterrupt |  
*  kENET_RxFrameInterrupt);  
*
```

Parameters

<i>base</i>	ENET peripheral base address.
<i>mask</i>	ENET interrupts to enable. This is a logical OR of the enumeration enet_interrupt_enable_t .

16.8.22 static void ENET_DisableInterrupts (ENET_Type * *base*, uint32_t *mask*) [inline], [static]

This function disables the ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [enet_interrupt_enable_t](#). For example, to disable the TX frame interrupt and RX frame interrupt, do the following.

```
*  ENET_DisableInterrupts(ENET, kENET_TxFrameInterrupt |  
*  kENET_RxFrameInterrupt);  
*
```

Parameters

<i>base</i>	ENET peripheral base address.
<i>mask</i>	ENET interrupts to disable. This is a logical OR of the enumeration enet_interrupt_enable_t .

16.8.23 static uint32_t ENET_GetInterruptStatus (ENET_Type * *base*) [inline], [static]

Parameters

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

Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration [enet_interrupt_enable_t](#).

16.8.24 static void ENET_ClearInterruptStatus (ENET_Type * *base*, uint32_t *mask*) [inline], [static]

This function clears enabled ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See the [enet_interrupt_enable_t](#). For example, to clear the TX frame interrupt and RX frame interrupt, do the following.

```
*  ENET_ClearInterruptStatus(ENET,  
    kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);  
*
```

Parameters

<i>base</i>	ENET peripheral base address.
<i>mask</i>	ENET interrupt source to be cleared. This is the logical OR of members of the enumeration enet_interrupt_enable_t .

16.8.25 void ENET_SetRxISRHandler (ENET_Type * *base*, enet_isr_t *ISRHandler*)

Function Documentation

Parameters

<i>base</i>	ENET peripheral base address.
<i>ISRHandler</i>	The handler to install.

16.8.26 void ENET_SetTxISRHandler (ENET_Type * *base*, enet_isr_t *ISRHandler*)

Parameters

<i>base</i>	ENET peripheral base address.
<i>ISRHandler</i>	The handler to install.

16.8.27 void ENET_SetErrISRHandler (ENET_Type * *base*, enet_isr_t *ISRHandler*)

Parameters

<i>base</i>	ENET peripheral base address.
<i>ISRHandler</i>	The handler to install.

16.8.28 void ENET_SetCallback (enet_handle_t * *handle*, enet_callback_t *callback*, void * *userData*)

This API is provided for the application callback required case when ENET interrupt is enabled. This API should be called after calling ENET_Init.

Parameters

<i>handle</i>	ENET handler pointer. Should be provided by application.
<i>callback</i>	The ENET callback function.
<i>userData</i>	The callback function parameter.

16.8.29 void ENET_GetRxErrBeforeReadFrame (enet_handle_t * *handle*, enet_data_error_stats_t * *eErrorStatic*, uint8_t *ringId*)

This API must be called after the ENET_GetRxFrameSize and before the [ENET_ReadFrame\(\)](#). If the ENET_GetRxFrameSize returns kStatus_ENET_RxFrameError, the ENET_GetRxErrBeforeReadFrame

can be used to get the exact error statistics. This is an example.

```
*      status = ENET_GetRxFrameSize(&g_handle, &length, 0);
*      if (status == kStatus_ENET_RxFrameError)
*      {
*          Comments: Get the error information of the received frame.
*          ENET_GetRxErrBeforeReadFrame(&g_handle, &eErrStatic, 0);
*          Comments: update the receive buffer.
*          ENET_ReadFrame(EXAMPLE_ENET, &g_handle, NULL, 0);
*      }
*
```

Parameters

<i>handle</i>	The ENET handler structure pointer. This is the same handler pointer used in the ENET_Init.
<i>eErrorStatic</i>	The error statistics structure pointer.
<i>ringId</i>	The ring index, range from 0 ~ FSL_FEATURE_ENET_QUEUE - 1.

16.8.30 status_t ENET_GetRxFrameSize (enet_handle_t * *handle*, uint32_t * *length*, uint8_t *ringId*)

This function gets a received frame size from the ENET buffer descriptors.

Note

The FCS of the frame is automatically removed by MAC and the size is the length without the FCS. After calling ENET_GetRxFrameSize, [ENET_ReadFrame\(\)](#) should be called to update the receive buffers if the result is not "kStatus_ENET_RxFrameEmpty".

Parameters

<i>handle</i>	The ENET handler structure. This is the same handler pointer used in the ENET_Init.
<i>length</i>	The length of the valid frame received.
<i>ringId</i>	The ring index or ring number.

Return values

<i>kStatus_ENET_RxFrame-Empty</i>	No frame received. Should not call ENET_ReadFrame to read frame.
-----------------------------------	--

Function Documentation

<i>kStatus_ENET_RxFrameError</i>	Data error happens. ENET_ReadFrame should be called with NULL data and NULL length to update the receive buffers.
<i>kStatus_Success</i>	Receive a frame Successfully then the ENET_ReadFrame should be called with the right data buffer and the captured data length input.

16.8.31 status_t ENET_ReadFrame (ENET_Type * *base*, enet_handle_t * *handle*, uint8_t * *data*, uint32_t *length*, uint8_t *ringld*, uint32_t * *ts*)

This function reads a frame (both the data and the length) from the ENET buffer descriptors. User can get timestamp through ts pointer if the ts is not NULL. Note that it doesn't store the timestamp in the receive timestamp queue. The ENET_GetRxFrameSize should be used to get the size of the prepared data buffer. This is an example:

```
*      uint32_t length;
*      enet_handle_t g_handle;
*      Comments: Get the received frame size firstly.
*      status = ENET_GetRxFrameSize(&g_handle, &length, 0);
*      if (length != 0)
*      {
*          Comments: Allocate memory here with the size of "length"
*          uint8_t *data = memory allocate interface;
*          if (!data)
*          {
*              ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0, NULL);
*              Comments: Add the console warning log.
*          }
*          else
*          {
*              status = ENET_ReadFrame(ENET, &g_handle, data, length, 0, NULL);
*              Comments: Call stack input API to deliver the data to stack
*          }
*      }
*      else if (status == kStatus_ENET_RxFrameError)
*      {
*          Comments: Update the received buffer when a error frame is received.
*          ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0, NULL);
*      }
*
```

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler structure. This is the same handler pointer used in the ENET_Init.
<i>data</i>	The data buffer provided by user to store the frame which memory size should be at least "length".

<i>length</i>	The size of the data buffer which is still the length of the received frame.
<i>ringId</i>	The ring index or ring number.
<i>ts</i>	The timestamp address to store received timestamp.

Returns

The execute status, successful or failure.

16.8.32 `status_t ENET_SendFrame (ENET_Type * base, enet_handle_t * handle, const uint8_t * data, uint32_t length, uint8_t ringId, bool tsFlag, void * context)`

Note

The CRC is automatically appended to the data. Input the data to send without the CRC.

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
<i>data</i>	The data buffer provided by user to send.
<i>length</i>	The length of the data to send.
<i>ringId</i>	The ring index or ring number.
<i>tsFlag</i>	Timestamp enable flag.
<i>context</i>	Used by user to handle some events after transmit over.

Return values

<i>kStatus_Success</i>	Send frame succeed.
<i>kStatus_ENET_TxFrame-Busy</i>	Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity. The waiting mechanism is recommended to be added after each call return with kStatus-ENET_TxFrameBusy.

16.8.33 `status_t ENET_SetTxReclaim (enet_handle_t * handle, bool isEnabled, uint8_t ringId)`

Function Documentation

Note

This function must be called when no pending send frame action. Set enable if you want to reclaim context or timestamp in interrupt.

Parameters

<i>handle</i>	The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
<i>isEnabled</i>	Enable or disable flag.
<i>ringId</i>	The ring index or ring number.

Return values

<i>kStatus_Success</i>	Succeed to enable/disable Tx reclaim.
<i>kStatus_Fail</i>	Fail to enable/disable Tx reclaim.

16.8.34 `status_t ENET_GetRxBuffer (ENET_Type * base, enet_handle_t * handle, void ** buffer, uint32_t * length, uint8_t ringId, bool * isLastBuff, uint32_t * ts)`

This function can get the data address which stores frame. Then can analyze these data directly without doing any memory copy. When the frame locates in multiple BD buffer, need to repeat calling this function until *isLastBuff*=true (need to store the temp buf pointer everytime call this function). After finishing the analysis of this frame, call `ENET_ReleaseRxBuffer` to release rxbuff memory to DMA. This is an example:

```
*      uint32_t length;
*      uint8_t *buf = NULL;
*      uint32_t data_len = 0;
*      bool isLastBuff = false;
*      enet_handle_t g_handle;
*      status_t status;
*      status = ENET_GetRxFrameSize(&g_handle, &length, 0);
*      if (length != 0)
*      {
*          ENET_GetRxBuffer(EXAMPLE_ENET, &g_handle, &buf, &data_len, 0, &isLastBuff, NULL
*      );
*          ENET_ReleaseRxBuffer(EXAMPLE_ENET, &g_handle, buf, 0);
*      }
*
```

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler structure. This is the same handler pointer used in the ENET_Init.
<i>buffer</i>	The data buffer pointer to store the frame.
<i>length</i>	The size of the data buffer. If isLastBuff=false, it represents data length of this buffer. If isLastBuff=true, it represents data length of total frame.
<i>ringId</i>	The ring index, range from 0 ~ FSL_FEATURE_ENET_QUEUE - 1.
<i>isLastBuff</i>	The flag represents whether this buffer is the last buffer to store frame.
<i>ts</i>	The 1588 timestamp value, valid in last buffer.

Return values

<i>kStatus_Success</i>	Get receive buffer succeed.
<i>kStatus_ENET_RxFrame-Fail</i>	Get receive buffer fails, it's owned by application, should wait app to release this buffer.

16.8.35 void ENET_ReleaseRxBuffer (ENET_Type * *base*, enet_handle_t * *handle*, void * *buffer*, uint8_t *ringId*)

This function can release specified BD owned by application, meanwhile it may rearrange the BD to let the no-owned BDs always in back of the index of DMA transfer. So for the situation that releasing order is not same as the getting order, the rearrangement makes all ready BDs can be used by DMA.

Note

This function can't be interrupted by ENET_GetRxBuffer, so in application must make sure ENET_GetRxBuffer is called before or after this function. And this function itself isn't thread safe due to BD content exchanging.

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler structure. This is the same handler pointer used in the ENET_Init.
<i>buffer</i>	The buffer address to store frame, using it to find the correspond BD and release it.

Function Documentation

<i>ringId</i>	The ring index, range from 0 ~ FSL_FEATURE_ENET_QUEUE - 1.
---------------	--

16.8.36 `status_t ENET_SendFrameZeroCopy (ENET_Type * base, enet_handle_t * handle, const uint8_t * data, uint32_t length, uint8_t ringId, bool tsFlag, void * context)`

Note

The CRC is automatically appended to the data. Input the data to send without the CRC. The frame must store in continuous memory and need to check the buffer start address alignment based on your device, otherwise it has issue or can't get highest DMA transmit speed.

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
<i>data</i>	The data buffer provided by user to send.
<i>length</i>	The length of the data to send.
<i>ringId</i>	The ring index or ring number.
<i>tsFlag</i>	Timestamp enable flag.
<i>context</i>	Used by user to handle some events after transmit over.

Return values

<i>kStatus_Success</i>	Send frame succeed.
<i>kStatus_ENET_TxFrame-Busy</i>	Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity. The waiting mechanism is recommended to be added after each call return with kStatus-ENET_TxFrameBusy.

16.8.37 `status_t ENET_SetTxBuffer (ENET_Type * base, enet_handle_t * handle, const uint8_t * data, uint32_t length, uint8_t ringId, uint8_t txFlag, void * context)`

This function only set one Tx BD everytime calls, all ready data will be sent out with last flag sets or gets error. Send frame succeeds with last flag sets, then you can get context from frameInfo in callback.

Note

The CRC is automatically appended to the data. Input the data to send without the CRC. And if doesn't succeed to call this function, user can't get context in frameInfo of callback.

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
<i>data</i>	The data buffer provided by user to send.
<i>length</i>	The length of the data to send.
<i>ringId</i>	The ring index, range from 0 ~ FSL_FEATURE_ENET_QUEUE - 1.
<i>txFlag</i>	This function uses timestamp enable flag, last BD flag.
<i>context</i>	Used by user to handle some events after transmit over.

Return values

<i>kStatus_Success</i>	Send frame succeed.
<i>kStatus_ENET_TxFrame-OverLen</i>	Buffer length isn't enough to store data.
<i>kStatus_ENET_TxFrame-Busy</i>	Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity.

16.8.38 void ENET_TransmitIRQHandler (ENET_Type * *base*, enet_handle_t * *handle*)

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler pointer.

16.8.39 void ENET_ReceiveIRQHandler (ENET_Type * *base*, enet_handle_t * *handle*)

Function Documentation

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler pointer.

16.8.40 void ENET_ErrorIRQHandler (ENET_Type * *base*, enet_handle_t * *handle*)

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler pointer.

16.8.41 void ENET_CommonFrame0IRQHandler (ENET_Type * *base*)

This is used for the combined tx/rx/error interrupt for single/multi-ring (frame 0).

Parameters

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

ENET CMSIS Driver

This section describes the programming interface of the ENET 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 ENET 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.

16.9.1 Typical use case

```
void ENET_SignalEvent_t(uint32_t event)
{
    if (event == ARM_ETH_MAC_EVENT_RX_FRAME)
    {
        uint32_t size;
        uint32_t len;

        /* Get the Frame size */
        size = EXAMPLE_ENET.GetRxFrameSize();
        /* Call ENET_ReadFrame when there is a received frame. */
        if (size != 0)
        {
            /* Received valid frame. Deliver the rx buffer with the size equal to length. */
            uint8_t *data = (uint8_t *)malloc(size);
            if (data)
            {
                len = EXAMPLE_ENET.ReadFrame(data, size);
                if (size == len)
                {
                    /* Increase the received frame numbers. */
                    if (g_rxIndex < ENET_EXAMPLE_LOOP_COUNT)
                    {
                        g_rxIndex++;
                    }
                }
                free(data);
            }
        }
    }
    if (event == ARM_ETH_MAC_EVENT_TX_FRAME)
    {
        g_testTxNum ++;
    }
}

/* Initialize the ENET module. */
EXAMPLE_ENET.Initialize(ENET_SignalEvent_t);
EXAMPLE_ENET.PowerControl(ARM_POWER_FULL);
EXAMPLE_ENET.SetMacAddress((ARM_ETH_MAC_ADDR *)g_macAddr);
EXAMPLE_ENET.Control(ARM_ETH_MAC_CONFIGURE, linkInfo.speed << ARM_ETH_MAC_SPEED_Pos |
    linkInfo.duplex << ARM_ETH_MAC_DUPLEX_Pos | ARM_ETH_MAC_ADDRESS_BROADCAST);
EXAMPLE_ENET_PHY.PowerControl(ARM_POWER_FULL);
```

ENET CMSIS Driver

```
EXAMPLE_ENET_PHY.SetMode(ARM_ETH_PHY_AUTO_NEGOTIATE);
EXAMPLE_ENET.Control(ARM_ETH_MAC_CONTROL_RX, 1);
EXAMPLE_ENET.Control(ARM_ETH_MAC_CONTROL_TX, 1);
if (EXAMPLE_ENET_PHY.GetLinkState() == ARM_ETH_LINK_UP)
{
    linkInfo = EXAMPLE_ENET_PHY.GetLinkInfo();
}
else
{
    PRINTF("\r\nPHY Link down, please check the cable connection and link partner setting.\r\n");
}

/* Build broadcast for sending. */
ENET_BuildBroadCastFrame();

while (1)
{
    /* Check the total number of received number. */
    if (g_rxCheckIdx != g_rxIndex)
    {
        PRINTF("The %d frame has been successfully received!\r\n", g_rxIndex);
        g_rxCheckIdx = g_rxIndex;
    }
    if (g_testTxNum && (g_txCheckIdx != g_testTxNum))
    {
        g_txCheckIdx = g_testTxNum;
        PRINTF("The %d frame transmitted success!\r\n", g_txCheckIdx);
    }
    /* Get the Frame size */
    if (txnumber < ENET_EXAMPLE_LOOP_COUNT)
    {
        txnumber++;
        /* Send a multicast frame when the PHY is link up. */
        if (EXAMPLE_ENET.SendFrame(&g_frame[0], ENET_DATA_LENGTH, ARM_ETH_MAC_TX_FRAME_EVENT) ==
            ARM_DRIVER_OK)
        {
            for (uint32_t count = 0; count < 0x3FF; count++)
            {
                __ASM("nop");
            }
        }
        else
        {
            PRINTF("\r\nTransmit frame failed!\r\n");
        }
    }
}
```

Chapter 17

EWM: External Watchdog Monitor Driver

Overview

The MCUXpresso SDK provides a peripheral driver for the External Watchdog (EWM) Driver module of MCUXpresso SDK devices.

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, 2))
EWM driver version 2.0.2.

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)
Serves the EWM.

Enumeration Type Documentation

Data Structure Documentation

17.3.1 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 [compareLowValue](#)
Compare low-register value.
- uint8_t [compareHighValue](#)
Compare high-register value.

Macro Definition Documentation

17.4.1 #define FSL_EWM_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

Enumeration Type Documentation

17.5.1 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.

17.5.2 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.

Function Documentation

17.6.1 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

17.6.2 void EWM_Deinit (EWM_Type * *base*)

This function is used to shut down the EWM.

Parameters

<i>base</i>	EWM peripheral base address
-------------	-----------------------------

17.6.3 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;
*
```

Function Documentation

Parameters

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

See Also

[ewm_config_t](#)

17.6.4 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

17.6.5 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

17.6.6 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.

17.6.7 void EWM_Refresh (EWM_Type * *base*)

This function resets the EWM counter to zero.

Parameters

<i>base</i>	EWM peripheral base address
-------------	-----------------------------

Chapter 18

C90TFS Flash Driver

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](#)

Ftftx FLASH Driver

18.2.1 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_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, 0U, 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_EraseAll \(flash_config_t *config, uint32_t key\)](#)
Erases entire flexnm.

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.
- [status_t FLASH_ProgramSection \(flash_config_t *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes\)](#)
Programs flash with data at locations passed in through parameters via the Program Section command.

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.

FlexRAM

- [status_t FLASH_SetFlexramFunction](#) ([flash_config_t](#) *config, [ftfx_flexram_func_opt_t](#) option)
Sets the FlexRAM function command.

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_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 FLASH_GetProperty](#) ([flash_config_t](#) *config, [flash_property_tag_t](#) whichProperty, [uint32_t](#) *value)
Returns the desired flash property.

18.2.2 Data Structure Documentation

18.2.2.1 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].

18.2.2.1.0.40 Field Documentation

18.2.2.1.0.40.1 uint32_t pflash_prot_status_t::protl

18.2.2.1.0.40.2 uint32_t pflash_prot_status_t::proth

18.2.2.1.0.40.3 uint8_t pflash_prot_status_t::protsl

18.2.2.1.0.40.4 uint8_t pflash_prot_status_t::protsh

18.2.2.2 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.

18.2.3 Macro Definition Documentation

18.2.3.1 #define FSL_FLASH_DRIVER_VERSION (MAKE_VERSION(3U, 0U, 2U))

Version 3.0.2.

18.2.3.2 #define FSL_FLASH_DRIVER_VERSION_ROM (MAKE_VERSION(3U, 0U, 0U))

Version 3.0.0.

18.2.4 Enumeration Type Documentation

18.2.4.1 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.

18.2.4.2 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.

18.2.5 Function Documentation

18.2.5.1 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.

18.2.5.2 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_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.

18.2.5.3 status_t FLASH_EraseAll (flash_config_t * config, uint32_t key)

Parameters

Ftftx FLASH Driver

<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_EraseKeyError</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.

18.2.5.4 **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.

18.2.5.5 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.

Ftftx FLASH Driver

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; The index indicating the area of program once field was programed 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.

18.2.5.6 `status_t FLASH_ProgramSection (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 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 programed successfully into flash based on start address and length.
-----------------------------	--

<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_SetFlexramAsRamError</i>	Failed to set flexram as RAM.
<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_RecoverFlexramAsEepromError</i>	Failed to recover FlexRAM as EEPROM.

18.2.5.7 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.

Ftftx FLASH Driver

<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.

18.2.5.8 **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.

18.2.5.9 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.

Ftftx FLASH Driver

<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.

18.2.5.10 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.
------------------------------------	--

18.2.5.11 **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 programed into specified 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.

Ftftx FLASH Driver

<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.

18.2.5.12 **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_Invalid-Argument</i>	An invalid argument is provided.

18.2.5.13 **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.

18.2.5.14 status_t FLASH_SetFlexramFunction (flash_config_t * config, ftfx_flexram_func_opt_t option)

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>option</i>	The option used to set the work mode of FlexRAM.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the FlexRAM has been successfully configured as RAM or EEPROM.
<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.
------------------------------------	--

18.2.5.15 **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.

18.2.5.16 **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_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during command execution.

18.2.5.17 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_InvalidArgument</i>	An invalid argument is provided.

18.2.5.18 status_t FLASH_GetProperty (flash_config_t * config, flash_property_tag_t whichProperty, uint32_t * value)

Ftftx FLASH Driver

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_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_Unknown-Property</i>	An unknown property tag.

Ftftx CACHE Driver

18.3.1 Overview

Data Structures

- struct [ftfx_prefetch_speculation_status_t](#)
FTFx prefetch speculation status. [More...](#)
- struct [ftfx_cache_config_t](#)
FTFx cache driver state information. [More...](#)

Enumerations

- enum [_ftfx_cache_ram_func_constants](#) { [kFTFx_CACHE_RamFuncMaxSizeInWords](#) = 16U }
Constants for execute-in-RAM flash function.

Functions

- [status_t FTFx_CACHE_Init](#) ([ftfx_cache_config_t](#) *config)
Initializes the global FTFx cache structure members.
- [status_t FTFx_CACHE_ClearCachePrefetchSpeculation](#) ([ftfx_cache_config_t](#) *config, bool isPre-Process)
Process the cache/prefetch/speculation to the flash.
- [status_t FTFx_CACHE_PflashSetPrefetchSpeculation](#) ([ftfx_prefetch_speculation_status_t](#) *speculation-Status)
Sets the PFlash prefetch speculation to the intended speculation status.
- [status_t FTFx_CACHE_PflashGetPrefetchSpeculation](#) ([ftfx_prefetch_speculation_status_t](#) *speculation-Status)
Gets the PFlash prefetch speculation status.

FTFx cache version

- #define [FSL_FTFX_CACHE_DRIVER_VERSION](#) ([MAKE_VERSION](#)(3, 0, 0))
Flexnvm driver version for SDK.

18.3.2 Data Structure Documentation

18.3.2.1 struct [ftfx_prefetch_speculation_status_t](#)

Data Fields

- bool [instructionOff](#)
Instruction speculation.
- bool [dataOff](#)

Ftftx CACHE Driver

Data speculation.

18.3.2.1.0.41 Field Documentation

18.3.2.1.0.41.1 bool ftfx_prefetch_speculation_status_t::instructionOff

18.3.2.1.0.41.2 bool ftfx_prefetch_speculation_status_t::dataOff

18.3.2.2 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.

18.3.2.2.0.42 Field Documentation

18.3.2.2.0.42.1 function_bit_operation_ptr_t ftfx_cache_config_t::bitOperFuncAddr

18.3.3 Macro Definition Documentation

18.3.3.1 #define FSL_FTFX_CACHE_DRIVER_VERSION (MAKE_VERSION(3, 0, 0))

Version 1.0.0.

18.3.4 Enumeration Type Documentation

18.3.4.1 enum _ftfx_cache_ram_func_constants

Enumerator

kFTFx_CACHE_RamFuncMaxSizeInWords The maximum size of execute-in-RAM function.

18.3.5 Function Documentation

18.3.5.1 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

<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.

18.3.5.2 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_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.

18.3.5.3 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
---------------------------	---

Ftftx CACHE Driver

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_Invalid-SpeculationOption</i>	An invalid speculation option argument is provided.

18.3.5.4 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.
-----------------------------	--------------------------------

Ftftx FLEXNVM Driver

18.4.1 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.

Flexnvm version

- #define [FSL_FLEXNVM_DRIVER_VERSION](#) ([MAKE_VERSION](#)(3, 0, 2))
Flexnvm driver version for SDK.

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_DflashProgramSection](#) ([flexnvm_config_t](#) *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)
Programs flash with data at locations passed in through parameters via the Program Section command.
- [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.

FlexRAM

- [status_t FLEXNVM_SetFlexramFunction](#) ([flexnvm_config_t](#) *config, [ftfx_flexram_func_opt_t](#) option)
Sets the FlexRAM function command.

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 FLEXNVM_GetProperty](#) ([flexnvm_config_t](#) *config, [flexnvm_property_tag_t](#) which-Property, [uint32_t](#) *value)
Returns the desired flexnvm property.

18.4.2 Data Structure Documentation

18.4.2.1 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.

18.4.3 Macro Definition Documentation

18.4.3.1 #define FSL_FLEXNVM_DRIVER_VERSION (MAKE_VERSION(3, 0, 2))

Version 3.0.2.

18.4.4 Enumeration Type Documentation

18.4.4.1 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.

18.4.5 Function Documentation

18.4.5.1 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.

18.4.5.2 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 data 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_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.

18.4.5.3 status_t FLEXNVM_EraseAll (flexnvm_config_t * config, uint32_t key)

Parameters

Ftftx FLEXNVM Driver

<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_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.

18.4.5.4 **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 data have been successfully programmed into specified 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_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.

18.4.5.5 status_t FLEXNVM_DflashProgramSection (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 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.

Ftftx FLEXNVM Driver

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the desired data have been successfully programmed into specified data flash area.
<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_SetFlexramAsRamError</i>	Failed to set flexram as RAM.
<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_RecoverFlexramAsEepromError</i>	Failed to recover FlexRAM as EEPROM.

18.4.5.6 `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_InvalidArgument</i>	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 command execution.

18.4.5.7 **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

Ftftx FLEXNVM Driver

<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.

18.4.5.8 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.

18.4.5.9 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.

Ftftx FLEXNVM Driver

<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.

18.4.5.10 `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 hve been programed 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_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.

18.4.5.11 **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.

18.4.5.12 **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.
---------------	--

Ftftx FLEXNVM Driver

<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.

18.4.5.13 **status_t FLEXNVM_SetFlexramFunction (flexnvm_config_t * *config*, ftfx_flexram_func_opt_t *option*)**

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>option</i>	The option used to set the work mode of FlexRAM.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the FlexRAM has been successfully configured as RAM or EEPROM
<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.

18.4.5.14 **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.

18.4.5.15 **status_t FLEXNVM_DflashSetProtection (flexnvm_config_t * *config*, uint8_t *protectStatus*)**

Ftftx FLEXNVM Driver

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_Invalid-Argument</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.

18.4.5.16 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_Invalid-Argument</i>	An invalid argument is provided.

<i>kStatus_FTFx_CommandNotSupported</i>	Flash API is not supported.
---	-----------------------------

18.4.5.17 **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.

18.4.5.18 **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.

Ftftx FLEXNVM Driver

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-CommandNotSupported</i>	Flash API is not supported.

18.4.5.19 status_t FLEXNVM_GetProperty (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_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_Unknown-Property</i>	An unknown property tag.

ftfx feature

18.5.1 Overview

Modules

- [ftfx 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.

18.5.2 Macro Definition Documentation

18.5.2.1 #define FTFx_DRIVER_IS_FLASH_RESIDENT 1U

Used for the flash resident application.

18.5.2.2 #define FTFx_DRIVER_IS_EXPORTED 0U

Used for the MCUXpresso SDK application.

18.5.2.3 #define FTFx_FLASH1_HAS_PROT_CONTROL (0U)

18.5.2.4 #define FTFx_FLASH1_HAS_XACC_CONTROL (0U)



ftfx feature

18.5.3 ftfx adapter

ftfx controller

18.6.1 Overview

Modules

- [ftfx utilities](#)

Data Structures

- struct [ftfx_spec_mem_t](#)
ftfx special memory access information. [More...](#)
- struct [ftfx_mem_desc_t](#)
Flash memory descriptor. [More...](#)
- struct [ftfx_ops_config_t](#)
Active FTFx information for the current operation. [More...](#)
- struct [ftfx_ifr_desc_t](#)
Flash IFR memory descriptor. [More...](#)
- struct [ftfx_config_t](#)
Flash driver state information. [More...](#)

Enumerations

- enum [ftfx_partition_flexram_load_opt_t](#) {
 [kFTFx_PartitionFlexramLoadOptLoadedWithValidEepromData](#),
 [kFTFx_PartitionFlexramLoadOptNotLoaded](#) = 0x01U }
Enumeration for the FlexRAM load during reset option.
- enum [ftfx_read_resource_opt_t](#) {
 [kFTFx_ResourceOptionFlashIfr](#),
 [kFTFx_ResourceOptionVersionId](#) = 0x01U }
Enumeration for the two possible options of flash read resource command.
- enum [ftfx_margin_value_t](#) {
 [kFTFx_MarginValueNormal](#),
 [kFTFx_MarginValueUser](#),
 [kFTFx_MarginValueFactory](#),
 [kFTFx_MarginValueInvalid](#) }
Enumeration for supported FTFx margin levels.
- enum [ftfx_security_state_t](#) {
 [kFTFx_SecurityStateNotSecure](#) = (int)0xc33cc33cu,
 [kFTFx_SecurityStateBackdoorEnabled](#) = (int)0x5aa55aa5u,
 [kFTFx_SecurityStateBackdoorDisabled](#) = (int)0x5ac33ca5u }
Enumeration for the three possible FTFx security states.
- enum [ftfx_flexram_func_opt_t](#) {
 [kFTFx_FlexramFuncOptAvailableAsRam](#) = 0xFFU,
 [kFTFx_FlexramFuncOptAvailableForEeprom](#) = 0x00U }
Enumeration for the two possible options of set FlexRAM function command.

ftfx controller

- 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(kStatusGroupFtfxDriver, 0),
 `kStatus_FTFx_AlignmentError`,
 `kStatus_FTFx_AddressError` = MAKE_STATUS(kStatusGroupFtfxDriver, 2),
 `kStatus_FTFx_AccessError`,
 `kStatus_FTFx_ProtectionViolation`,
 `kStatus_FTFx_CommandFailure`,
 `kStatus_FTFx_UnknownProperty` = MAKE_STATUS(kStatusGroupFtfxDriver, 6),
 `kStatus_FTFx_EraseKeyError` = MAKE_STATUS(kStatusGroupFtfxDriver, 7),
 `kStatus_FTFx_RegionExecuteOnly` = MAKE_STATUS(kStatusGroupFtfxDriver, 8),
 `kStatus_FTFx_ExecuteInRamFunctionNotReady`,
 `kStatus_FTFx_PartitionStatusUpdateFailure`,
 `kStatus_FTFx_SetFlexramAsEepromError`,
 `kStatus_FTFx_RecoverFlexramAsRamError`,
 `kStatus_FTFx_SetFlexramAsRamError` = MAKE_STATUS(kStatusGroupFtfxDriver, 13),
 `kStatus_FTFx_RecoverFlexramAsEepromError`,
 `kStatus_FTFx_CommandNotSupported` = MAKE_STATUS(kStatusGroupFtfxDriver, 15),
 `kStatus_FTFx_SwapSystemNotInUninitialized`,
 `kStatus_FTFx_SwapIndicatorAddressError`,
 `kStatus_FTFx_ReadOnlyProperty` = MAKE_STATUS(kStatusGroupFtfxDriver, 18),
 `kStatus_FTFx_InvalidPropertyValue`,
 `kStatus_FTFx_InvalidSpeculationOption` }
 FTFx driver status codes.
- #define `kStatusGroupGeneric` 0
 FTFx driver status group.
- #define `kStatusGroupFtfxDriver` 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.
- status_t `FTFx_API_UpdateFlexnvmPartitionStatus` (`ftfx_config_t` *config)
Updates FlexNVM memory partition status according to data flash 0 IFR.

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_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.
- status_t `FTFx_CMD_ProgramSection` (`ftfx_config_t` *config, uint32_t start, const uint8_t *src, uint32_t lengthInBytes)
Programs flash with data at locations passed in through parameters via the Program Section command.
- status_t `FTFx_CMD_ProgramPartition` (`ftfx_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 `FTFx_CMD_ReadOnce` (`ftfx_config_t` *config, uint32_t index, uint8_t *dst, uint32_t lengthInBytes)
Reads the Program Once Field through parameters.

ftfx controller

- [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, [ftfx_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, [ftfx_margin_value_t](#) margin)
Verifies erasure of the entire flash at a specified margin level.
- [status_t FTFx_CMD_VerifyEraseAllExecuteOnlySegments](#) ([ftfx_config_t](#) *config, [ftfx_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, [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 FTFx_REG_GetSecurityState](#) ([ftfx_config_t](#) *config, [ftfx_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.

FlexRAM

- [status_t FTFx_CMD_SetFlexramFunction](#) ([ftfx_config_t](#) *config, [ftfx_flexram_func_opt_t](#) option)
Sets the FlexRAM function command.

18.6.2 Data Structure Documentation

18.6.2.1 struct [ftfx_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.

18.6.2.1.0.43 Field Documentation**18.6.2.1.0.43.1 uint32_t ftfx_spec_mem_t::base****18.6.2.1.0.43.2 uint32_t ftfx_spec_mem_t::size****18.6.2.1.0.43.3 uint32_t ftfx_spec_mem_t::count****18.6.2.2 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.

18.6.2.2.0.44 Field Documentation**18.6.2.2.0.44.1 uint8_t ftfx_mem_desc_t::type****18.6.2.2.0.44.2 uint8_t ftfx_mem_desc_t::index****18.6.2.2.0.44.3 uint32_t ftfx_mem_desc_t::totalSize****18.6.2.2.0.44.4 uint32_t ftfx_mem_desc_t::sectorSize****18.6.2.2.0.44.5 uint32_t ftfx_mem_desc_t::blockCount****18.6.2.3 struct ftfx_ops_config_t****Data Fields**

- uint32_t [convertedAddress](#)
A converted address for the current flash type.

ftfx controller

18.6.2.3.0.45 Field Documentation

18.6.2.3.0.45.1 uint32_t ftfx_ops_config_t::convertedAddress

18.6.2.4 struct ftfx_ifr_desc_t

18.6.2.5 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.

18.6.2.5.0.46 Field Documentation

18.6.2.5.0.46.1 function_ptr_t ftfx_config_t::runCmdFuncAddr

18.6.3 Macro Definition Documentation

18.6.3.1 #define kStatusGroupGeneric 0

18.6.4 Enumeration Type Documentation

18.6.4.1 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.

18.6.4.2 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.

18.6.4.3 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.

18.6.4.4 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.

18.6.4.5 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.

18.6.4.6 enum ftfx_security_state_t

Enumerator

kFTFx_SecurityStateNotSecure Flash is not secure.

kFTFx_SecurityStateBackdoorEnabled Flash backdoor is enabled.

kFTFx_SecurityStateBackdoorDisabled Flash backdoor is disabled.

18.6.4.7 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.

18.6.4.8 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.

18.6.5 Function Documentation

18.6.5.1 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.
---------------	--

18.6.5.2 status_t FTFx_API_UpdateFlexnvmPartitionStatus (ftfx_config_t * *config*)

This function updates FlexNVM memory partition status.

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_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_Partition-StatusUpdateFailure</i>	Failed to update the partition status.

18.6.5.3 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

ftfx controller

<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.

18.6.5.4 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.

18.6.5.5 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.

ftfx controller

<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.
------------------------------------	--

18.6.5.6 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.
------------------------------------	--

18.6.5.7 **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.

18.6.5.8 **status_t FTFx_CMD_ProgramSection (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 length.

ftfx controller

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 specified baseline.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.
<i>kStatus_FTFx_SetFlexramAsRamError</i>	Failed to set flexram as RAM.
<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_RecoverFlexramAsEepromError</i>	Failed to recover FlexRAM as EEPROM.

18.6.5.9 `status_t FTFx_CMD_ProgramPartition (ftfx_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>eeepromData-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.
<i>kStatus_FTFx_InvalidArgument</i>	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 command execution.

18.6.5.10 `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.

ftfx controller

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.

18.6.5.11 `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.

18.6.5.12 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.

ftfx controller

<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.

18.6.5.13 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_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.

18.6.5.14 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_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.

18.6.5.15 `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.

ftfx controller

<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.

18.6.5.16 `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 back-door 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_InvalidArgument</i>	An invalid argument is provided.

18.6.5.17 **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_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.

18.6.5.18 **status_t FTFx_CMD_SetFlexramFunction (ftfx_config_t * *config*, ftfx_flexram_func_opt_t *option*)**

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>option</i>	The option used to set the work mode of FlexRAM.

ftfx controller

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.

18.6.6 ftfx utilities

18.6.6.1 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 **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.
- #define **B1P4**(b) (((uint32_t)(b)&0xFFU) << 24U)
bytes2word utility.

18.6.6.2 Macro Definition Documentation

18.6.6.2.1 #define MAKE_VERSION(major, minor, bugfix) (((major) << 16) | ((minor) << 8) | (bugfix))

18.6.6.2.2 #define MAKE_STATUS(group, code) (((group)*100) + (code))

18.6.6.2.3 #define FOUR_CHAR_CODE(a, b, c, d) (((uint32_t)(d) << 24u) | ((uint32_t)(c) << 16u) | ((uint32_t)(b) << 8u) | ((uint32_t)(a)))

18.6.6.2.4 #define ALIGN_DOWN(x, a) (((uint32_t)(x)) & ~((uint32_t)(a)-1u))

18.6.6.2.5 #define ALIGN_UP(x, a) ALIGN_DOWN((uint32_t)(x) + (uint32_t)(a)-1u, a)

18.6.6.2.6 #define B1P4(b) (((uint32_t)(b)&0xFFU) << 24U)

Chapter 19

FlexBus: External Bus Interface Driver

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.

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.

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,

Typical use case and example

- ```
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\_TSI0 = 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\_TSI1 = 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.*

## Data Structure Documentation

### 19.4.1 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`

## Enumeration Type Documentation

- Byte-lane shift enable.*
- [flexbus\\_write\\_address\\_hold\\_t](#) writeAddressHold  
*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.*

## Macro Definition Documentation

### 19.5.1 #define FSL\_FLEXBUS\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))

## Enumeration Type Documentation

### 19.6.1 enum flexbus\_port\_size\_t

Enumerator

- kFLEXBUS\_4Bytes* 32-bit port size
- kFLEXBUS\_1Byte* 8-bit port size
- kFLEXBUS\_2Bytes* 16-bit port size

### 19.6.2 enum flexbus\_write\_address\_hold\_t

Enumerator

- kFLEXBUS\_Hold1Cycle* Hold address and attributes one cycles after FB\_CS<sub>n</sub> negates on writes.
- kFLEXBUS\_Hold2Cycles* Hold address and attributes two cycles after FB\_CS<sub>n</sub> negates on writes.
- kFLEXBUS\_Hold3Cycles* Hold address and attributes three cycles after FB\_CS<sub>n</sub> negates on writes.
- kFLEXBUS\_Hold4Cycles* Hold address and attributes four cycles after FB\_CS<sub>n</sub> negates on writes.

### 19.6.3 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.

#### 19.6.4 enum flexbus\_address\_setup\_t

Enumerator

***kFLEXBUS\_FirstRisingEdge*** Assert FB\_CS<sub>n</sub> on first rising clock edge after address is asserted.

***kFLEXBUS\_SecondRisingEdge*** Assert FB\_CS<sub>n</sub> on second rising clock edge after address is asserted.

***kFLEXBUS\_ThirdRisingEdge*** Assert FB\_CS<sub>n</sub> on third rising clock edge after address is asserted.

***kFLEXBUS\_FourthRisingEdge*** Assert FB\_CS<sub>n</sub> on fourth rising clock edge after address is asserted.

#### 19.6.5 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

#### 19.6.6 enum flexbus\_multiplex\_group1\_t

Enumerator

***kFLEXBUS\_MultiplexGroup1\_FB\_ALE*** FB\_ALE.

***kFLEXBUS\_MultiplexGroup1\_FB\_CS1*** FB\_CS1.

***kFLEXBUS\_MultiplexGroup1\_FB\_TS*** FB\_TS.

#### 19.6.7 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.

## Function Documentation

### 19.6.8 enum flexbus\_multiplex\_group3\_t

Enumerator

*kFLEXBUS\_MultiplexGroup3\_FB\_CS5* FB\_CS5.  
*kFLEXBUS\_MultiplexGroup3\_FB\_TSI1* FB\_TSI1.  
*kFLEXBUS\_MultiplexGroup3\_FB\_BE\_23\_16* FB\_BE\_23\_16.

### 19.6.9 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.

### 19.6.10 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.

## Function Documentation

### 19.7.1 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 |

**19.7.2 void FLEXBUS\_Deinit ( FB\_Type \* *base* )**

This function disables the clock gate of the FlexBus module clock.

## Parameters

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

**19.7.3 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 20

# FlexCAN: Flex Controller Area Network Driver

### Overview

The MCUXpresso SDK provides a peripheral driver for the Flex Controller Area Network (FlexCAN) module of MCUXpresso SDK devices.

### Modules

- [FlexCAN Driver](#)

## FlexCAN Driver

### 20.2.1 Overview

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

### 20.2.2 Typical use case

#### 20.2.2.1 Message Buffer Send Operation

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

#### 20.2.2.2 Message Buffer Receive Operation

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

#### 20.2.2.3 Receive FIFO Operation

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

## Data Structures

- struct [flexcan\\_frame\\_t](#)  
*FlexCAN message frame structure. [More...](#)*
- struct [flexcan\\_timing\\_config\\_t](#)  
*FlexCAN protocol timing characteristic configuration structure. [More...](#)*
- struct [flexcan\\_config\\_t](#)  
*FlexCAN module configuration structure. [More...](#)*
- struct [flexcan\\_rx\\_mb\\_config\\_t](#)  
*FlexCAN Receive Message Buffer configuration structure. [More...](#)*
- struct [flexcan\\_rx\\_fifo\\_config\\_t](#)  
*FlexCAN Rx FIFO configuration structure. [More...](#)*
- struct [flexcan\\_mb\\_transfer\\_t](#)  
*FlexCAN Message Buffer transfer. [More...](#)*
- struct [flexcan\\_fifo\\_transfer\\_t](#)  
*FlexCAN Rx FIFO transfer. [More...](#)*
- struct [flexcan\\_handle\\_t](#)  
*FlexCAN handle structure. [More...](#)*



## Macros

- #define **FLEXCAN\_ID\_STD**(id) (((uint32\_t)((uint32\_t)(id)) << CAN\_ID\_STD\_SHIFT)) & CAN\_ID\_STD\_MASK)  
*FlexCAN Frame ID helper macro.*
- #define **FLEXCAN\_ID\_EXT**(id)  
*Extend Frame ID helper macro.*
- #define **FLEXCAN\_RX\_MB\_STD\_MASK**(id, rtr, ide)  
*FlexCAN Rx Message Buffer Mask helper macro.*
- #define **FLEXCAN\_RX\_MB\_EXT\_MASK**(id, rtr, ide)  
*Extend Rx Message Buffer Mask helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_A**(id, rtr, ide)  
*FlexCAN Rx FIFO Mask helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_B\_HIGH**(id, rtr, ide)  
*Standard Rx FIFO Mask helper macro Type B upper part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_B\_LOW**(id, rtr, ide)  
*Standard Rx FIFO Mask helper macro Type B lower part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_HIGH**(id) (((uint32\_t)(id)&0x7F8) << 21)  
*Standard Rx FIFO Mask helper macro Type C upper part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_HIGH**(id) (((uint32\_t)(id)&0x7F8) << 13)  
*Standard Rx FIFO Mask helper macro Type C mid-upper part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_LOW**(id) (((uint32\_t)(id)&0x7F8) << 5)  
*Standard Rx FIFO Mask helper macro Type C mid-lower part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_LOW**(id) (((uint32\_t)(id)&0x7F8) >> 3)  
*Standard Rx FIFO Mask helper macro Type C lower part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_A**(id, rtr, ide)  
*Extend Rx FIFO Mask helper macro Type A helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_HIGH**(id, rtr, ide)  
*Extend Rx FIFO Mask helper macro Type B upper part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_LOW**(id, rtr, ide)  
*Extend Rx FIFO Mask helper macro Type B lower part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_HIGH**(id) ((FLEXCAN\_ID\_EXT(id) & 0x1FE00000) << 3)  
*Extend Rx FIFO Mask helper macro Type C upper part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_HIGH**(id)  
*Extend Rx FIFO Mask helper macro Type C mid-upper part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_LOW**(id)  
*Extend Rx FIFO Mask helper macro Type C mid-lower part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_LOW**(id) ((FLEXCAN\_ID\_EXT(id) & 0x1FE00000) >> 21)  
*Extend Rx FIFO Mask helper macro Type C lower part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_A**(id, rtr, ide) **FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_A**(id, rtr, ide)  
*FlexCAN Rx FIFO Filter helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_HIGH**(id, rtr, ide)  
*Standard Rx FIFO Filter helper macro Type B upper part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_LOW**(id, rtr, ide)

- *Standard Rx FIFO Filter helper macro Type B lower part helper macro.*  
• #define `FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH(id)`
- *Standard Rx FIFO Filter helper macro Type C upper part helper macro.*  
• #define `FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH(id)`
- *Standard Rx FIFO Filter helper macro Type C mid-upper part helper macro.*  
• #define `FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW(id)`
- *Standard Rx FIFO Filter helper macro Type C mid-lower part helper macro.*  
• #define `FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW(id)`
- *Standard Rx FIFO Filter helper macro Type C lower part helper macro.*  
• #define `FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A(id, rtr, ide) FLEXCAN_RX_FIFO_EXT-_MASK_TYPE_A(id, rtr, ide)`
- *Extend Rx FIFO Filter helper macro Type A helper macro.*  
• #define `FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH(id, rtr, ide)`
- *Extend Rx FIFO Filter helper macro Type B upper part helper macro.*  
• #define `FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW(id, rtr, ide)`
- *Extend Rx FIFO Filter helper macro Type B lower part helper macro.*  
• #define `FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH(id)`
- *Extend Rx FIFO Filter helper macro Type C upper part helper macro.*  
• #define `FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH(id)`
- *Extend Rx FIFO Filter helper macro Type C mid-upper part helper macro.*  
• #define `FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW(id)`
- *Extend Rx FIFO Filter helper macro Type C mid-lower part helper macro.*  
• #define `FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW(id) FLEXCAN_RX_FIFO_EXT-_MASK_TYPE_C_LOW(id)`
- *Extend Rx FIFO Filter helper macro Type C lower part helper macro.*

## Typedefs

- typedef void(\* `flexcan_transfer_callback_t`)(CAN\_Type \*base, flexcan\_handle\_t \*handle, `status_t` status, uint32\_t result, void \*userData)  
*FlexCAN transfer callback function.*

## Enumerations

- enum {
  - kStatus\_FLEXCAN\_TxBusy = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 0),
  - kStatus\_FLEXCAN\_TxIdle = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 1),
  - kStatus\_FLEXCAN\_TxSwitchToRx,
  - kStatus\_FLEXCAN\_RxBusy = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 3),
  - kStatus\_FLEXCAN\_RxIdle = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 4),
  - kStatus\_FLEXCAN\_RxOverflow = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 5),
  - kStatus\_FLEXCAN\_RxFifoBusy = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 6),
  - kStatus\_FLEXCAN\_RxFifoIdle = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 7),
  - kStatus\_FLEXCAN\_RxFifoOverflow = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 8),
  - kStatus\_FLEXCAN\_RxFifoWarning = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 9),
  - kStatus\_FLEXCAN\_ErrorStatus = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 10),
  - kStatus\_FLEXCAN\_WakeUp = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 11),
  - kStatus\_FLEXCAN\_UnHandled = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 12),
  - kStatus\_FLEXCAN\_RxRemote = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 13) }

*FlexCAN transfer status.*
- enum flexcan\_frame\_format\_t {
  - kFLEXCAN\_FrameFormatStandard = 0x0U,
  - kFLEXCAN\_FrameFormatExtend = 0x1U }

*FlexCAN frame format.*
- enum flexcan\_frame\_type\_t {
  - kFLEXCAN\_FrameTypeData = 0x0U,
  - kFLEXCAN\_FrameTypeRemote = 0x1U }

*FlexCAN frame type.*
- enum flexcan\_clock\_source\_t {
  - kFLEXCAN\_ClkSrcOsc = 0x0U,
  - kFLEXCAN\_ClkSrcPeri = 0x1U,
  - kFLEXCAN\_ClkSrc0 = 0x0U,
  - kFLEXCAN\_ClkSrc1 = 0x1U }

*FlexCAN clock source.*
- enum flexcan\_wake\_up\_source\_t {
  - kFLEXCAN\_WakeupSrcUnfiltered = 0x0U,
  - kFLEXCAN\_WakeupSrcFiltered = 0x1U }

*FlexCAN wake up source.*
- enum flexcan\_rx\_fifo\_filter\_type\_t {
  - kFLEXCAN\_RxFifoFilterTypeA = 0x0U,
  - kFLEXCAN\_RxFifoFilterTypeB,
  - kFLEXCAN\_RxFifoFilterTypeC,
  - kFLEXCAN\_RxFifoFilterTypeD = 0x3U }

*FlexCAN Rx Fifo Filter type.*
- enum flexcan\_rx\_fifo\_priority\_t {
  - kFLEXCAN\_RxFifoPrioLow = 0x0U,
  - kFLEXCAN\_RxFifoPrioHigh = 0x1U }

*FlexCAN Rx FIFO priority.*
- enum \_flexcan\_interrupt\_enable {

## FlexCAN Driver

```
kFLEXCAN_BusOffInterruptEnable = CAN_CTRL1_BOFFMSK_MASK,
kFLEXCAN_ErrorInterruptEnable = CAN_CTRL1_ERRMSK_MASK,
kFLEXCAN_RxWarningInterruptEnable = CAN_CTRL1_RWRNMSK_MASK,
kFLEXCAN_TxWarningInterruptEnable = CAN_CTRL1_TWRNMSK_MASK,
kFLEXCAN_WakeUpInterruptEnable = CAN_MCR_WAKMSK_MASK }
```

*FlexCAN interrupt configuration structure, default settings all disabled.*

- enum `_flexcan_flags` {  
    kFLEXCAN\_SynchFlag = CAN\_ESR1\_SYNCH\_MASK,  
    kFLEXCAN\_TxWarningIntFlag = CAN\_ESR1\_TWRNINT\_MASK,  
    kFLEXCAN\_RxWarningIntFlag = CAN\_ESR1\_RWRNINT\_MASK,  
    kFLEXCAN\_TxErrorWarningFlag = CAN\_ESR1\_TXWRN\_MASK,  
    kFLEXCAN\_RxErrorWarningFlag = CAN\_ESR1\_RXWRN\_MASK,  
    kFLEXCAN\_IdleFlag = CAN\_ESR1\_IDLE\_MASK,  
    kFLEXCAN\_FaultConfinementFlag = CAN\_ESR1\_FLTCONF\_MASK,  
    kFLEXCAN\_TransmittingFlag = CAN\_ESR1\_TX\_MASK,  
    kFLEXCAN\_ReceivingFlag = CAN\_ESR1\_RX\_MASK,  
    kFLEXCAN\_BusOffIntFlag = CAN\_ESR1\_BOFFINT\_MASK,  
    kFLEXCAN\_ErrorIntFlag = CAN\_ESR1\_ERRINT\_MASK,  
    kFLEXCAN\_WakeUpIntFlag = CAN\_ESR1\_WAKINT\_MASK }

*FlexCAN status flags.*

- enum `_flexcan_error_flags` {  
    kFLEXCAN\_StuffingError = CAN\_ESR1\_STFERR\_MASK,  
    kFLEXCAN\_FormError = CAN\_ESR1\_FRMERR\_MASK,  
    kFLEXCAN\_CrcError = CAN\_ESR1\_CRCERR\_MASK,  
    kFLEXCAN\_AckError = CAN\_ESR1\_ACKERR\_MASK,  
    kFLEXCAN\_Bit0Error = CAN\_ESR1\_BIT0ERR\_MASK,  
    kFLEXCAN\_Bit1Error = CAN\_ESR1\_BIT1ERR\_MASK }

*FlexCAN error status flags.*

- enum {  
    kFLEXCAN\_RxFifoOverflowFlag = CAN\_IFLAG1\_BUF7I\_MASK,  
    kFLEXCAN\_RxFifoWarningFlag = CAN\_IFLAG1\_BUF6I\_MASK,  
    kFLEXCAN\_RxFifoFrameAvlFlag = CAN\_IFLAG1\_BUF5I\_MASK }

*FlexCAN Rx FIFO status flags.*

## Driver version

- #define `FSL_FLEXCAN_DRIVER_VERSION` (`MAKE_VERSION(2, 5, 2)`)  
    *FlexCAN driver version.*

## Initialization and deinitialization

- void `FLEXCAN_EnterFreezeMode` (CAN\_Type \*base)  
    *Enter FlexCAN Freeze Mode.*
- void `FLEXCAN_ExitFreezeMode` (CAN\_Type \*base)  
    *Exit FlexCAN Freeze Mode.*

- `uint32_t FLEXCAN_GetInstance (CAN_Type *base)`  
*Get the FlexCAN instance from peripheral base address.*
- `bool FLEXCAN_CalculateImprovedTimingValues (uint32_t baudRate, uint32_t sourceClock_Hz, flexcan_timing_config_t *pTimingConfig)`  
*Calculates the improved timing values by specific baudrates for classical CAN.*
- `void FLEXCAN_Init (CAN_Type *base, const flexcan_config_t *pConfig, uint32_t sourceClock_Hz)`  
*Initializes a FlexCAN instance.*
- `void FLEXCAN_Deinit (CAN_Type *base)`  
*De-initializes a FlexCAN instance.*
- `void FLEXCAN_GetDefaultConfig (flexcan_config_t *pConfig)`  
*Gets the default configuration structure.*

## Configuration.

- `void FLEXCAN_SetTimingConfig (CAN_Type *base, const flexcan_timing_config_t *pConfig)`  
*Sets the FlexCAN protocol timing characteristic.*
- `void FLEXCAN_SetRxMbGlobalMask (CAN_Type *base, uint32_t mask)`  
*Sets the FlexCAN receive message buffer global mask.*
- `void FLEXCAN_SetRxFifoGlobalMask (CAN_Type *base, uint32_t mask)`  
*Sets the FlexCAN receive FIFO global mask.*
- `void FLEXCAN_SetRxIndividualMask (CAN_Type *base, uint8_t maskIdx, uint32_t mask)`  
*Sets the FlexCAN receive individual mask.*
- `void FLEXCAN_SetTxMbConfig (CAN_Type *base, uint8_t mbIdx, bool enable)`  
*Configures a FlexCAN transmit message buffer.*
- `void FLEXCAN_SetRxMbConfig (CAN_Type *base, uint8_t mbIdx, const flexcan_rx_mb_config_t *pRxMbConfig, bool enable)`  
*Configures a FlexCAN Receive Message Buffer.*
- `void FLEXCAN_SetRxFifoConfig (CAN_Type *base, const flexcan_rx_fifo_config_t *pRxFifoConfig, bool enable)`  
*Configures the FlexCAN Rx FIFO.*

## Status

- `static uint32_t FLEXCAN_GetStatusFlags (CAN_Type *base)`  
*Gets the FlexCAN module interrupt flags.*
- `static void FLEXCAN_ClearStatusFlags (CAN_Type *base, uint32_t mask)`  
*Clears status flags with the provided mask.*
- `static void FLEXCAN_GetBusErrCount (CAN_Type *base, uint8_t *txErrBuf, uint8_t *rxErrBuf)`  
*Gets the FlexCAN Bus Error Counter value.*
- `static uint32_t FLEXCAN_GetMbStatusFlags (CAN_Type *base, uint32_t mask)`  
*Gets the FlexCAN Message Buffer interrupt flags.*
- `static void FLEXCAN_ClearMbStatusFlags (CAN_Type *base, uint32_t mask)`  
*Clears the FlexCAN Message Buffer interrupt flags.*

### Interrupts

- static void [FLEXCAN\\_EnableInterrupts](#) (CAN\_Type \*base, uint32\_t mask)  
*Enables FlexCAN interrupts according to the provided mask.*
- static void [FLEXCAN\\_DisableInterrupts](#) (CAN\_Type \*base, uint32\_t mask)  
*Disables FlexCAN interrupts according to the provided mask.*
- static void [FLEXCAN\\_EnableMbInterrupts](#) (CAN\_Type \*base, uint32\_t mask)  
*Enables FlexCAN Message Buffer interrupts.*
- static void [FLEXCAN\\_DisableMbInterrupts](#) (CAN\_Type \*base, uint32\_t mask)  
*Disables FlexCAN Message Buffer interrupts.*

### Bus Operations

- static void [FLEXCAN\\_Enable](#) (CAN\_Type \*base, bool enable)  
*Enables or disables the FlexCAN module operation.*
- [status\\_t FLEXCAN\\_WriteTxMb](#) (CAN\_Type \*base, uint8\_t mbIdx, const [flexcan\\_frame\\_t](#) \*pTxFrame)  
*Writes a FlexCAN Message to the Transmit Message Buffer.*
- [status\\_t FLEXCAN\\_ReadRxMb](#) (CAN\_Type \*base, uint8\_t mbIdx, [flexcan\\_frame\\_t](#) \*pRxFrame)  
*Reads a FlexCAN Message from Receive Message Buffer.*
- [status\\_t FLEXCAN\\_ReadRx Fifo](#) (CAN\_Type \*base, [flexcan\\_frame\\_t](#) \*pRxFrame)  
*Reads a FlexCAN Message from Rx FIFO.*

### Transactional

- [status\\_t FLEXCAN\\_TransferSendBlocking](#) (CAN\_Type \*base, uint8\_t mbIdx, [flexcan\\_frame\\_t](#) \*pTxFrame)  
*Performs a polling send transaction on the CAN bus.*
- [status\\_t FLEXCAN\\_TransferReceiveBlocking](#) (CAN\_Type \*base, uint8\_t mbIdx, [flexcan\\_frame\\_t](#) \*pRxFrame)  
*Performs a polling receive transaction on the CAN bus.*
- [status\\_t FLEXCAN\\_TransferReceiveFifoBlocking](#) (CAN\_Type \*base, [flexcan\\_frame\\_t](#) \*pRxFrame)  
*Performs a polling receive transaction from Rx FIFO on the CAN bus.*
- void [FLEXCAN\\_TransferCreateHandle](#) (CAN\_Type \*base, [flexcan\\_handle\\_t](#) \*handle, [flexcan\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the FlexCAN handle.*
- [status\\_t FLEXCAN\\_TransferSendNonBlocking](#) (CAN\_Type \*base, [flexcan\\_handle\\_t](#) \*handle, [flexcan\\_mb\\_transfer\\_t](#) \*pMbXfer)  
*Sends a message using IRQ.*
- [status\\_t FLEXCAN\\_TransferReceiveNonBlocking](#) (CAN\_Type \*base, [flexcan\\_handle\\_t](#) \*handle, [flexcan\\_mb\\_transfer\\_t](#) \*pMbXfer)  
*Receives a message using IRQ.*
- [status\\_t FLEXCAN\\_TransferReceiveFifoNonBlocking](#) (CAN\_Type \*base, [flexcan\\_handle\\_t](#) \*handle, [flexcan\\_fifo\\_transfer\\_t](#) \*pFifoXfer)  
*Receives a message from Rx FIFO using IRQ.*
- uint32\_t [FLEXCAN\\_GetTimeStamp](#) ([flexcan\\_handle\\_t](#) \*handle, uint8\_t mbIdx)

- Gets the detail index of Mailbox's Timestamp by handle.*
- void **FLEXCAN\_TransferAbortSend** (CAN\_Type \*base, flexcan\_handle\_t \*handle, uint8\_t mbIdx)  
*Aborts the interrupt driven message send process.*
- void **FLEXCAN\_TransferAbortReceive** (CAN\_Type \*base, flexcan\_handle\_t \*handle, uint8\_t mbIdx)  
*Aborts the interrupt driven message receive process.*
- void **FLEXCAN\_TransferAbortReceiveFifo** (CAN\_Type \*base, flexcan\_handle\_t \*handle)  
*Aborts the interrupt driven message receive from Rx FIFO process.*
- void **FLEXCAN\_TransferHandleIRQ** (CAN\_Type \*base, flexcan\_handle\_t \*handle)  
*FlexCAN IRQ handle function.*



### 20.2.3 Data Structure Documentation

#### 20.2.3.1 struct flexcan\_frame\_t

##### 20.2.3.1.0.1 Field Documentation

20.2.3.1.0.1.1 uint32\_t flexcan\_frame\_t::timestamp

20.2.3.1.0.1.2 uint32\_t flexcan\_frame\_t::length

20.2.3.1.0.1.3 uint32\_t flexcan\_frame\_t::type

20.2.3.1.0.1.4 uint32\_t flexcan\_frame\_t::format

20.2.3.1.0.1.5 uint32\_t flexcan\_frame\_t::\_\_pad0\_\_

20.2.3.1.0.1.6 uint32\_t flexcan\_frame\_t::idhit

20.2.3.1.0.1.7 uint32\_t flexcan\_frame\_t::id

20.2.3.1.0.1.8 uint32\_t flexcan\_frame\_t::dataWord0

20.2.3.1.0.1.9 uint32\_t flexcan\_frame\_t::dataWord1

20.2.3.1.0.1.10 uint8\_t flexcan\_frame\_t::dataByte3

20.2.3.1.0.1.11 uint8\_t flexcan\_frame\_t::dataByte2

20.2.3.1.0.1.12 uint8\_t flexcan\_frame\_t::dataByte1

20.2.3.1.0.1.13 uint8\_t flexcan\_frame\_t::dataByte0

20.2.3.1.0.1.14 uint8\_t flexcan\_frame\_t::dataByte7

20.2.3.1.0.1.15 uint8\_t flexcan\_frame\_t::dataByte6

20.2.3.1.0.1.16 uint8\_t flexcan\_frame\_t::dataByte5

20.2.3.1.0.1.17 uint8\_t flexcan\_frame\_t::dataByte4

#### 20.2.3.2 struct flexcan\_timing\_config\_t

##### Data Fields

- uint16\_t [preDivider](#)  
*Clock Pre-scaler Division Factor.*
- uint8\_t [rJumpwidth](#)  
*Re-sync Jump Width.*
- uint8\_t [phaseSeg1](#)  
*Phase Segment 1.*



- uint8\_t [phaseSeg2](#)  
*Phase Segment 2.*
- uint8\_t [propSeg](#)  
*Propagation Segment.*

#### 20.2.3.2.0.2 Field Documentation

20.2.3.2.0.2.1 uint16\_t flexcan\_timing\_config\_t::preDivider

20.2.3.2.0.2.2 uint8\_t flexcan\_timing\_config\_t::rJumpwidth

20.2.3.2.0.2.3 uint8\_t flexcan\_timing\_config\_t::phaseSeg1

20.2.3.2.0.2.4 uint8\_t flexcan\_timing\_config\_t::phaseSeg2

20.2.3.2.0.2.5 uint8\_t flexcan\_timing\_config\_t::propSeg

#### 20.2.3.3 struct flexcan\_config\_t

##### Data Fields

- uint32\_t [baudRate](#)  
*FlexCAN baud rate in bps.*
- [flexcan\\_clock\\_source\\_t](#) clkSrc  
*Clock source for FlexCAN Protocol Engine.*
- [flexcan\\_wake\\_up\\_source\\_t](#) wakeupSrc  
*Wake up source selection.*
- uint8\_t [maxMbNum](#)  
*The maximum number of Message Buffers used by user.*
- bool [enableLoopBack](#)  
*Enable or Disable Loop Back Self Test Mode.*
- bool [enableTimerSync](#)  
*Enable or Disable Timer Synchronization.*
- bool [enableSelfWakeup](#)  
*Enable or Disable Self Wakeup Mode.*
- bool [enableIndividMask](#)  
*Enable or Disable Rx Individual Mask.*
- bool [disableSelfReception](#)  
*Enable or Disable Self Reflection.*
- bool [enableListenOnlyMode](#)  
*Enable or Disable Listen Only Mode.*

### 20.2.3.3.0.3 Field Documentation

20.2.3.3.0.3.1 `uint32_t flexcan_config_t::baudRate`

20.2.3.3.0.3.2 `flexcan_clock_source_t flexcan_config_t::clkSrc`

20.2.3.3.0.3.3 `flexcan_wake_up_source_t flexcan_config_t::wakeupSrc`

20.2.3.3.0.3.4 `uint8_t flexcan_config_t::maxMbNum`

20.2.3.3.0.3.5 `bool flexcan_config_t::enableLoopBack`

20.2.3.3.0.3.6 `bool flexcan_config_t::enableTimerSync`

20.2.3.3.0.3.7 `bool flexcan_config_t::enableSelfWakeup`

20.2.3.3.0.3.8 `bool flexcan_config_t::enableIndividMask`

20.2.3.3.0.3.9 `bool flexcan_config_t::disableSelfReception`

20.2.3.3.0.3.10 `bool flexcan_config_t::enableListenOnlyMode`

### 20.2.3.4 `struct flexcan_rx_mb_config_t`

This structure is used as the parameter of [FLEXCAN\\_SetRxMbConfig\(\)](#) function. The [FLEXCAN\\_SetRxMbConfig\(\)](#) function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

### Data Fields

- `uint32_t id`  
*CAN Message Buffer Frame Identifier, should be set using [FLEXCAN\\_ID\\_EXT\(\)](#) or [FLEXCAN\\_ID\\_STD\(\)](#) macro.*
- `flexcan_frame_format_t format`  
*CAN Frame Identifier format(Standard of Extend).*
- `flexcan_frame_type_t type`  
*CAN Frame Type(Data or Remote).*

#### 20.2.3.4.0.4 Field Documentation

20.2.3.4.0.4.1 `uint32_t flexcan_rx_mb_config_t::id`

20.2.3.4.0.4.2 `flexcan_frame_format_t flexcan_rx_mb_config_t::format`

20.2.3.4.0.4.3 `flexcan_frame_type_t flexcan_rx_mb_config_t::type`

#### 20.2.3.5 struct `flexcan_rx_fifo_config_t`

##### Data Fields

- `uint32_t * idFilterTable`  
*Pointer to the FlexCAN Rx FIFO identifier filter table.*
- `uint8_t idFilterNum`  
*The quantity of filter elements.*
- `flexcan_rx_fifo_filter_type_t idFilterType`  
*The FlexCAN Rx FIFO Filter type.*
- `flexcan_rx_fifo_priority_t priority`  
*The FlexCAN Rx FIFO receive priority.*

#### 20.2.3.5.0.5 Field Documentation

20.2.3.5.0.5.1 `uint32_t* flexcan_rx_fifo_config_t::idFilterTable`

20.2.3.5.0.5.2 `uint8_t flexcan_rx_fifo_config_t::idFilterNum`

20.2.3.5.0.5.3 `flexcan_rx_fifo_filter_type_t flexcan_rx_fifo_config_t::idFilterType`

20.2.3.5.0.5.4 `flexcan_rx_fifo_priority_t flexcan_rx_fifo_config_t::priority`

#### 20.2.3.6 struct `flexcan_mb_transfer_t`

##### Data Fields

- `flexcan_frame_t * frame`  
*The buffer of CAN Message to be transfer.*
- `uint8_t mbIdx`  
*The index of Message buffer used to transfer Message.*

#### 20.2.3.6.0.6 Field Documentation

20.2.3.6.0.6.1 `flexcan_frame_t* flexcan_mb_transfer_t::frame`

20.2.3.6.0.6.2 `uint8_t flexcan_mb_transfer_t::mbIdx`

#### 20.2.3.7 struct `flexcan_fifo_transfer_t`

##### Data Fields

- `flexcan_frame_t * frame`

## FlexCAN Driver

*The buffer of CAN Message to be received from Rx FIFO.*

### 20.2.3.7.0.7 Field Documentation

#### 20.2.3.7.0.7.1 flexcan\_frame\_t\* flexcan\_fifo\_transfer\_t::frame

### 20.2.3.8 struct \_flexcan\_handle

FlexCAN handle structure definition.

#### Data Fields

- [flexcan\\_transfer\\_callback\\_t](#) callback  
*Callback function.*
- void \* [userData](#)  
*FlexCAN callback function parameter.*
- [flexcan\\_frame\\_t](#) \*volatile [mbFrameBuf](#) [CAN\_WORD1\_COUNT]  
*The buffer for received data from Message Buffers.*
- [flexcan\\_frame\\_t](#) \*volatile [rxFifoFrameBuf](#)  
*The buffer for received data from Rx FIFO.*
- volatile uint8\_t [mbState](#) [CAN\_WORD1\_COUNT]  
*Message Buffer transfer state.*
- volatile uint8\_t [rxFifoState](#)  
*Rx FIFO transfer state.*
- volatile uint32\_t [timestamp](#) [CAN\_WORD1\_COUNT]  
*Mailbox transfer timestamp.*

#### 20.2.3.8.0.8 Field Documentation

20.2.3.8.0.8.1 `flexcan_transfer_callback_t flexcan_handle_t::callback`

20.2.3.8.0.8.2 `void* flexcan_handle_t::userData`

20.2.3.8.0.8.3 `flexcan_frame_t* volatile flexcan_handle_t::mbFrameBuf[CAN_WORD1_COUNT]`

20.2.3.8.0.8.4 `flexcan_frame_t* volatile flexcan_handle_t::rxFifoFrameBuf`

20.2.3.8.0.8.5 `volatile uint8_t flexcan_handle_t::mbState[CAN_WORD1_COUNT]`

20.2.3.8.0.8.6 `volatile uint8_t flexcan_handle_t::rxFifoState`

20.2.3.8.0.8.7 `volatile uint32_t flexcan_handle_t::timestamp[CAN_WORD1_COUNT]`

#### 20.2.4 Macro Definition Documentation

20.2.4.1 `#define FSL_FLEXCAN_DRIVER_VERSION (MAKE_VERSION(2, 5, 2))`

20.2.4.2 `#define FLEXCAN_ID_STD( id ) (((uint32_t)((uint32_t)(id)) << CAN_ID_STD_SHIFT)) & CAN_ID_STD_MASK)`

Standard Frame ID helper macro.

20.2.4.3 `#define FLEXCAN_ID_EXT( id )`

**Value:**

```
((uint32_t)((uint32_t)(id)) << CAN_ID_EXT_SHIFT)) & \
(CAN_ID_EXT_MASK | CAN_ID_STD_MASK)
```

20.2.4.4 `#define FLEXCAN_RX_MB_STD_MASK( id, rtr, ide )`

**Value:**

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
FLEXCAN_ID_STD(id))
```

Standard Rx Message Buffer Mask helper macro.

20.2.4.5 `#define FLEXCAN_RX_MB_EXT_MASK( id, rtr, ide )`

**Value:**

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
FLEXCAN_ID_EXT(id))
```

### 20.2.4.6 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_A( *id*, *rtr*, *ide* )

**Value:**

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
(FLEXCAN_ID_STD(id) << 1)
```

Standard Rx FIFO Mask helper macro Type A helper macro.

### 20.2.4.7 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )

**Value:**

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
((uint32_t)(id)&0x7FF) << 19)
```

### 20.2.4.8 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_B\_LOW( *id*, *rtr*, *ide* )

**Value:**

```
((uint32_t)((uint32_t)(rtr) << 15) | (uint32_t)((uint32_t)(ide) << 14)) | \
((uint32_t)(id)&0x7FF) << 3)
```

### 20.2.4.9 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_HIGH( *id* ) (((uint32\_t)(id)&0x7F8) << 21)

### 20.2.4.10 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_HIGH( *id* ) (((uint32\_t)(id)&0x7F8) << 13)

### 20.2.4.11 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_LOW( *id* ) (((uint32\_t)(id)&0x7F8) << 5)

### 20.2.4.12 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_LOW( *id* ) (((uint32\_t)(id)&0x7F8) >> 3)

### 20.2.4.13 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_A( *id*, *rtr*, *ide* )

**Value:**

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
(FLEXCAN_ID_EXT(id) << 1)
```

**20.2.4.14 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )****Value:**

```
(
 ((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
 ((FLEXCAN_ID_EXT(id) & 0x1FFF8000) << 1))
```

**20.2.4.15 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_LOW( *id*, *rtr*, *ide* )****Value:**

```
((uint32_t)((uint32_t)(rtr) << 15) | (uint32_t)((uint32_t)(ide) << 14)) | \
 ((FLEXCAN_ID_EXT(id) & 0x1FFF8000) >> 15))
```

**20.2.4.16 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_HIGH( *id* ) ((FLEXCAN\_ID\_EXT(id) & 0x1FE00000) << 3)****20.2.4.17 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_HIGH( *id* )****Value:**

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 5)
```

**20.2.4.18 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_LOW( *id* )****Value:**

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 13)
```

**20.2.4.19 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_LOW( *id* ) ((FLEXCAN\_ID\_EXT(id) & 0x1FE00000) >> 21)****20.2.4.20 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_A( *id*, *rtr*, *ide* ) FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_A(id, rtr, ide)**

Standard Rx FIFO Filter helper macro Type A helper macro.

### 20.2.4.21 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(\
 id, rtr, ide)
```

### 20.2.4.22 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_LOW( *id*, *rtr*, *ide* )

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(\
 id, rtr, ide)
```

### 20.2.4.23 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_HIGH( *id* )

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(\
 id)
```

### 20.2.4.24 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_MID\_HIGH( *id* )

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(\
 id)
```

### 20.2.4.25 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_MID\_LOW( *id* )

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(\
 id)
```

### 20.2.4.26 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_LOW( *id* )

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(\
 id)
```



**20.2.4.27 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_A( *id*, *rtr*, *ide*  
 ) FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_A(id, rtr, ide)**

**20.2.4.28 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(\
 id, rtr, ide)
```

**20.2.4.29 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_B\_LOW( *id*, *rtr*, *ide* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(\
 id, rtr, ide)
```

**20.2.4.30 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_HIGH( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(\
 id)
```

**20.2.4.31 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_MID\_HIGH( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(\
 id)
```

**20.2.4.32 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_MID\_LOW( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(\
 id)
```

**20.2.4.33** `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW( id  
 ) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)`

### 20.2.5 Typedef Documentation

**20.2.5.1** `typedef void(* flexcan_transfer_callback_t)(CAN_Type *base, flexcan_handle_t  
*handle, status_t status, uint32_t result, void *userData)`

The FlexCAN transfer callback returns a value from the underlying layer. If the status equals to `kStatus_FLEXCAN_ErrorStatus`, the result parameter is the Content of FlexCAN status register which can be used to get the working status(or error status) of FlexCAN module. If the status equals to other FlexCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other FlexCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

### 20.2.6 Enumeration Type Documentation

#### 20.2.6.1 anonymous enum

Enumerator

*kStatus\_FLEXCAN\_TxBusy* Tx Message Buffer is Busy.  
*kStatus\_FLEXCAN\_TxIdle* Tx Message Buffer is Idle.  
*kStatus\_FLEXCAN\_TxSwitchToRx* Remote Message is send out and Message buffer changed to Receive one.  
*kStatus\_FLEXCAN\_RxBusy* Rx Message Buffer is Busy.  
*kStatus\_FLEXCAN\_RxIdle* Rx Message Buffer is Idle.  
*kStatus\_FLEXCAN\_RxOverflow* Rx Message Buffer is Overflowed.  
*kStatus\_FLEXCAN\_RxFifoBusy* Rx Message FIFO is Busy.  
*kStatus\_FLEXCAN\_RxFifoIdle* Rx Message FIFO is Idle.  
*kStatus\_FLEXCAN\_RxFifoOverflow* Rx Message FIFO is overflowed.  
*kStatus\_FLEXCAN\_RxFifoWarning* Rx Message FIFO is almost overflowed.  
*kStatus\_FLEXCAN\_ErrorStatus* FlexCAN Module Error and Status.  
*kStatus\_FLEXCAN\_WakeUp* FlexCAN is waken up from STOP mode.  
*kStatus\_FLEXCAN\_UnHandled* UnHandled Interrupt asserted.  
*kStatus\_FLEXCAN\_RxRemote* Rx Remote Message Received in Mail box.

#### 20.2.6.2 enum flexcan\_frame\_format\_t

Enumerator

*kFLEXCAN\_FrameFormatStandard* Standard frame format attribute.  
*kFLEXCAN\_FrameFormatExtend* Extend frame format attribute.

### 20.2.6.3 enum flexcan\_frame\_type\_t

Enumerator

***kFLEXCAN\_FrameTypeData*** Data frame type attribute.

***kFLEXCAN\_FrameTypeRemote*** Remote frame type attribute.

### 20.2.6.4 enum flexcan\_clock\_source\_t

**Deprecated** Do not use the `kFLEXCAN_ClkSrcOs`. It has been superceded `kFLEXCAN_ClkSrc0`  
Do not use the `kFLEXCAN_ClkSrcPeri`. It has been superceded `kFLEXCAN_ClkSrc1`

Enumerator

***kFLEXCAN\_ClkSrcOsc*** FlexCAN Protocol Engine clock from Oscillator.

***kFLEXCAN\_ClkSrcPeri*** FlexCAN Protocol Engine clock from Peripheral Clock.

***kFLEXCAN\_ClkSrc0*** FlexCAN Protocol Engine clock selected by user as SRC == 0.

***kFLEXCAN\_ClkSrc1*** FlexCAN Protocol Engine clock selected by user as SRC == 1.

### 20.2.6.5 enum flexcan\_wake\_up\_source\_t

Enumerator

***kFLEXCAN\_WakeupSrcUnfiltered*** FlexCAN uses unfiltered Rx input to detect edge.

***kFLEXCAN\_WakeupSrcFiltered*** FlexCAN uses filtered Rx input to detect edge.

### 20.2.6.6 enum flexcan\_rx\_fifo\_filter\_type\_t

Enumerator

***kFLEXCAN\_RxFifoFilterTypeA*** One full ID (standard and extended) per ID Filter element.

***kFLEXCAN\_RxFifoFilterTypeB*** Two full standard IDs or two partial 14-bit ID slices per ID Filter Table element.

***kFLEXCAN\_RxFifoFilterTypeC*** Four partial 8-bit Standard or extended ID slices per ID Filter Table element.

***kFLEXCAN\_RxFifoFilterTypeD*** All frames rejected.

### 20.2.6.7 enum flexcan\_rx\_fifo\_priority\_t

The matching process starts from the Rx MB(or Rx FIFO) with higher priority. If no MB(or Rx FIFO filter) is satisfied, the matching process goes on with the Rx FIFO(or Rx MB) with lower priority.

### Enumerator

***kFLEXCAN\_RxFifoPrioLow*** Matching process start from Rx Message Buffer first.

***kFLEXCAN\_RxFifoPrioHigh*** Matching process start from Rx FIFO first.

### 20.2.6.8 enum \_flexcan\_interrupt\_enable

This structure contains the settings for all of the FlexCAN Module interrupt configurations. Note: FlexCAN Message Buffers and Rx FIFO have their own interrupts.

### Enumerator

***kFLEXCAN\_BusOffInterruptEnable*** Bus Off interrupt.

***kFLEXCAN\_ErrorInterruptEnable*** Error interrupt.

***kFLEXCAN\_RxWarningInterruptEnable*** Rx Warning interrupt.

***kFLEXCAN\_TxWarningInterruptEnable*** Tx Warning interrupt.

***kFLEXCAN\_WakeUpInterruptEnable*** Wake Up interrupt.

### 20.2.6.9 enum \_flexcan\_flags

This provides constants for the FlexCAN status flags for use in the FlexCAN functions. Note: The CPU read action clears FLEXCAN\_ErrorFlag, therefore user need to read FLEXCAN\_ErrorFlag and distinguish which error is occur using [\\_flexcan\\_error\\_flags](#) enumerations.

### Enumerator

***kFLEXCAN\_SynchFlag*** CAN Synchronization Status.

***kFLEXCAN\_TxWarningIntFlag*** Tx Warning Interrupt Flag.

***kFLEXCAN\_RxWarningIntFlag*** Rx Warning Interrupt Flag.

***kFLEXCAN\_TxErrorWarningFlag*** Tx Error Warning Status.

***kFLEXCAN\_RxErrorWarningFlag*** Rx Error Warning Status.

***kFLEXCAN\_IdleFlag*** CAN IDLE Status Flag.

***kFLEXCAN\_FaultConfinementFlag*** Fault Confinement State Flag.

***kFLEXCAN\_TransmittingFlag*** FlexCAN In Transmission Status.

***kFLEXCAN\_ReceivingFlag*** FlexCAN In Reception Status.

***kFLEXCAN\_BusOffIntFlag*** Bus Off Interrupt Flag.

***kFLEXCAN\_ErrorIntFlag*** Error Interrupt Flag.

***kFLEXCAN\_WakeUpIntFlag*** Wake-Up Interrupt Flag.

### 20.2.6.10 enum \_flexcan\_error\_flags

The FlexCAN Error Status enumerations is used to report current error of the FlexCAN bus. This enumerations should be used with KFlexCAN\_ErrorFlag in [\\_flexcan\\_flags](#) enumerations to determine which error is generated.

## Enumerator

***kFLEXCAN\_StuffingError*** Stuffing Error.  
***kFLEXCAN\_FormError*** Form Error.  
***kFLEXCAN\_CrcError*** Cyclic Redundancy Check Error.  
***kFLEXCAN\_AckError*** Received no ACK on transmission.  
***kFLEXCAN\_Bit0Error*** Unable to send dominant bit.  
***kFLEXCAN\_Bit1Error*** Unable to send recessive bit.

### 20.2.6.11 anonymous enum

The FlexCAN Rx FIFO Status enumerations are used to determine the status of the Rx FIFO. Because Rx FIFO occupy the MB0 ~ MB7 (Rx Fifo filter also occupies more Message Buffer space), Rx FIFO status flags are mapped to the corresponding Message Buffer status flags.

## Enumerator

***kFLEXCAN\_RxFifoOverflowFlag*** Rx FIFO overflow flag.  
***kFLEXCAN\_RxFifoWarningFlag*** Rx FIFO almost full flag.  
***kFLEXCAN\_RxFifoFrameAvlFlag*** Frames available in Rx FIFO flag.

## 20.2.7 Function Documentation

### 20.2.7.1 void FLEXCAN\_EnterFreezeMode ( CAN\_Type \* *base* )

This function makes the FlexCAN work under Freeze Mode.

Parameters

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

### 20.2.7.2 void FLEXCAN\_ExitFreezeMode ( CAN\_Type \* *base* )

This function makes the FlexCAN leave Freeze Mode.

Parameters

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

### 20.2.7.3 uint32\_t FLEXCAN\_GetInstance ( CAN\_Type \* *base* )

## FlexCAN Driver

### Parameters

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

### Returns

FlexCAN instance.

#### 20.2.7.4 **bool FLEXCAN\_CalculateImprovedTimingValues ( uint32\_t *baudRate*, uint32\_t *sourceClock\_Hz*, flexcan\_timing\_config\_t \* *pTimingConfig* )**

### Parameters

|                       |                                                                        |
|-----------------------|------------------------------------------------------------------------|
| <i>baudRate</i>       | The classical CAN speed in bps defined by user                         |
| <i>sourceClock_Hz</i> | The Source clock data speed in bps. Zero to disable baudrate switching |
| <i>pTimingConfig</i>  | Pointer to the FlexCAN timing configuration structure.                 |

### Returns

TRUE if timing configuration found, FALSE if failed to find configuration

#### 20.2.7.5 **void FLEXCAN\_Init ( CAN\_Type \* *base*, const flexcan\_config\_t \* *pConfig*, uint32\_t *sourceClock\_Hz* )**

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the [flexcan\\_config\\_t](#) parameters and how to call the FLEXCAN\_Init function by passing in these parameters.

```
* flexcan_config_t flexcanConfig;
* flexcanConfig.clkSrc = kFLEXCAN_ClkSrc0;
* flexcanConfig.baudRate = 1000000U;
* flexcanConfig.maxMbNum = 16;
* flexcanConfig.enableLoopBack = false;
* flexcanConfig.enableSelfWakeup = false;
* flexcanConfig.enableIndividMask = false;
* flexcanConfig.enableDoze = false;
* flexcanConfig.disableSelfReception = false;
* flexcanConfig.enableListenOnlyMode = false;
* flexcanConfig.timingConfig = timingConfig;
* FLEXCAN_Init(CAN0, &flexcanConfig, 8000000UL);
*
```

## Parameters

|                                  |                                                       |
|----------------------------------|-------------------------------------------------------|
| <i>base</i>                      | FlexCAN peripheral base address.                      |
| <i>pConfig</i>                   | Pointer to the user-defined configuration structure.  |
| <i>sourceClock_</i><br><i>Hz</i> | FlexCAN Protocol Engine clock source frequency in Hz. |

**20.2.7.6 void FLEXCAN\_Deinit ( CAN\_Type \* *base* )**

This function disables the FlexCAN module clock and sets all register values to the reset value.

## Parameters

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

**20.2.7.7 void FLEXCAN\_GetDefaultConfig ( flexcan\_config\_t \* *pConfig* )**

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. flexcanConfig->clkSrc = kFLEXCAN\_ClkSrc0; flexcanConfig->baudRate = 1000000U; flexcanConfig->baudRateFD = 2000000U; flexcanConfig->maxMbNum = 16; flexcanConfig->enableLoopBack = false; flexcanConfig->enableSelfWakeup = false; flexcanConfig->enableIndividMask = false; flexcanConfig->disableSelfReception = false; flexcanConfig->enableListenOnlyMode = false; flexcanConfig->enableDoze = false; flexcanConfig.timingConfig = timingConfig;

## Parameters

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

**20.2.7.8 void FLEXCAN\_SetTimingConfig ( CAN\_Type \* *base*, const flexcan\_timing\_config\_t \* *pConfig* )**

This function gives user settings to CAN bus timing characteristic. The function is for an experienced user. For less experienced users, call the [FLEXCAN\\_Init\(\)](#) and fill the baud rate field with a desired value. This provides the default timing characteristics to the module.

Note that calling [FLEXCAN\\_SetTimingConfig\(\)](#) overrides the baud rate set in [FLEXCAN\\_Init\(\)](#).

## FlexCAN Driver

### Parameters

|                |                                                |
|----------------|------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.               |
| <i>pConfig</i> | Pointer to the timing configuration structure. |

#### 20.2.7.9 void FLEXCAN\_SetRxMbGlobalMask ( CAN\_Type \* *base*, uint32\_t *mask* )

This function sets the global mask for the FlexCAN message buffer in a matching process. The configuration is only effective when the Rx individual mask is disabled in the [FLEXCAN\\_Init\(\)](#).

### Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.     |
| <i>mask</i> | Rx Message Buffer Global Mask value. |

#### 20.2.7.10 void FLEXCAN\_SetRxFifoGlobalMask ( CAN\_Type \* *base*, uint32\_t *mask* )

This function sets the global mask for FlexCAN FIFO in a matching process.

### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FlexCAN peripheral base address. |
| <i>mask</i> | Rx Fifo Global Mask value.       |

#### 20.2.7.11 void FLEXCAN\_SetRxIndividualMask ( CAN\_Type \* *base*, uint8\_t *maskIdx*, uint32\_t *mask* )

This function sets the individual mask for the FlexCAN matching process. The configuration is only effective when the Rx individual mask is enabled in the [FLEXCAN\\_Init\(\)](#). If the Rx FIFO is disabled, the individual mask is applied to the corresponding Message Buffer. If the Rx FIFO is enabled, the individual mask for Rx FIFO occupied Message Buffer is applied to the Rx Filter with the same index. Note that only the first 32 individual masks can be used as the Rx FIFO filter mask.

### Parameters

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



|                |                               |
|----------------|-------------------------------|
| <i>maskIdx</i> | The Index of individual Mask. |
| <i>mask</i>    | Rx Individual Mask value.     |

#### 20.2.7.12 void FLEXCAN\_SetTxMbConfig ( CAN\_Type \* *base*, uint8\_t *mbIdx*, bool *enable* )

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

|               |                                                                                                                                                                    |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                                                                                                   |
| <i>mbIdx</i>  | The Message Buffer index.                                                                                                                                          |
| <i>enable</i> | Enable/disable Tx Message Buffer. <ul style="list-style-type: none"> <li>• true: Enable Tx Message Buffer.</li> <li>• false: Disable Tx Message Buffer.</li> </ul> |

#### 20.2.7.13 void FLEXCAN\_SetRxMbConfig ( CAN\_Type \* *base*, uint8\_t *mbIdx*, const flexcan\_rx\_mb\_config\_t \* *pRxMbConfig*, bool *enable* )

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

Parameters

|                    |                                                                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | FlexCAN peripheral base address.                                                                                                                                   |
| <i>mbIdx</i>       | The Message Buffer index.                                                                                                                                          |
| <i>pRxMbConfig</i> | Pointer to the FlexCAN Message Buffer configuration structure.                                                                                                     |
| <i>enable</i>      | Enable/disable Rx Message Buffer. <ul style="list-style-type: none"> <li>• true: Enable Rx Message Buffer.</li> <li>• false: Disable Rx Message Buffer.</li> </ul> |

#### 20.2.7.14 void FLEXCAN\_SetRxFifoConfig ( CAN\_Type \* *base*, const flexcan\_rx\_fifo\_config\_t \* *pRxFifoConfig*, bool *enable* )

This function configures the Rx FIFO with given Rx FIFO configuration.

## FlexCAN Driver

### Parameters

|                      |                                                                                                                                   |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | FlexCAN peripheral base address.                                                                                                  |
| <i>pRxFifoConfig</i> | Pointer to the FlexCAN Rx FIFO configuration structure.                                                                           |
| <i>enable</i>        | Enable/disable Rx FIFO. <ul style="list-style-type: none"><li>• true: Enable Rx FIFO.</li><li>• false: Disable Rx FIFO.</li></ul> |

### 20.2.7.15 static uint32\_t FLEXCAN\_GetStatusFlags ( CAN\_Type \* *base* ) [inline], [static]

This function gets all FlexCAN status flags. The flags are returned as the logical OR value of the enumerators [\\_flexcan\\_flags](#). To check the specific status, compare the return value with enumerators in [\\_flexcan\\_flags](#).

### Parameters

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

### Returns

FlexCAN status flags which are ORed by the enumerators in the [\\_flexcan\\_flags](#).

### 20.2.7.16 static void FLEXCAN\_ClearStatusFlags ( CAN\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function clears the FlexCAN status flags with a provided mask. An automatically cleared flag can't be cleared by this function.

### Parameters

|             |                                                                                            |
|-------------|--------------------------------------------------------------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.                                                           |
| <i>mask</i> | The status flags to be cleared, it is logical OR value of <a href="#">_flexcan_flags</a> . |

### 20.2.7.17 static void FLEXCAN\_GetBusErrCount ( CAN\_Type \* *base*, uint8\_t \* *txErrBuf*, uint8\_t \* *rxErrBuf* ) [inline], [static]

This function gets the FlexCAN Bus Error Counter value for both Tx and Rx direction. These values may be needed in the upper layer error handling.

## Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.        |
| <i>txErrBuf</i> | Buffer to store Tx Error Counter value. |
| <i>rxErrBuf</i> | Buffer to store Rx Error Counter value. |

#### 20.2.7.18 static uint32\_t FLEXCAN\_GetMbStatusFlags ( CAN\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function gets the interrupt flags of a given Message Buffers.

## Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

## Returns

The status of given Message Buffers.

#### 20.2.7.19 static void FLEXCAN\_ClearMbStatusFlags ( CAN\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function clears the interrupt flags of a given Message Buffers.

## Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

#### 20.2.7.20 static void FLEXCAN\_EnableInterrupts ( CAN\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function enables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see [\\_flexcan\\_interrupt\\_enable](#).

## FlexCAN Driver

### Parameters

|             |                                                                                     |
|-------------|-------------------------------------------------------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.                                                    |
| <i>mask</i> | The interrupts to enable. Logical OR of <a href="#">_flexcan_interrupt_enable</a> . |

#### 20.2.7.21 **static void FLEXCAN\_DisableInterrupts ( CAN\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

This function disables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see [\\_flexcan\\_interrupt\\_enable](#).

### Parameters

|             |                                                                                      |
|-------------|--------------------------------------------------------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.                                                     |
| <i>mask</i> | The interrupts to disable. Logical OR of <a href="#">_flexcan_interrupt_enable</a> . |

#### 20.2.7.22 **static void FLEXCAN\_EnableMblInterrupts ( CAN\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

This function enables the interrupts of given Message Buffers.

### Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

#### 20.2.7.23 **static void FLEXCAN\_DisableMblInterrupts ( CAN\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

This function disables the interrupts of given Message Buffers.

### Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

#### 20.2.7.24 **static void FLEXCAN\_Enable ( CAN\_Type \* *base*, bool *enable* ) [inline], [static]**

This function enables or disables the FlexCAN module.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FlexCAN base pointer.             |
| <i>enable</i> | true to enable, false to disable. |

### 20.2.7.25 **status\_t FLEXCAN\_WriteTxMb ( CAN\_Type \* *base*, uint8\_t *mbIdx*, const flexcan\_frame\_t \* *pTxFrame* )**

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

## Parameters

|                 |                                          |
|-----------------|------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.         |
| <i>mbIdx</i>    | The FlexCAN Message Buffer index.        |
| <i>pTxFrame</i> | Pointer to CAN message frame to be sent. |

## Return values

|                        |                                          |
|------------------------|------------------------------------------|
| <i>kStatus_Success</i> | - Write Tx Message Buffer Successfully.  |
| <i>kStatus_Fail</i>    | - Tx Message Buffer is currently in use. |

### 20.2.7.26 **status\_t FLEXCAN\_ReadRxMb ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_frame\_t \* *pRxFrame* )**

This function reads a CAN message from a specified Receive Message Buffer. The function fills a receive CAN message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

## Parameters

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.                      |
| <i>mbIdx</i>    | The FlexCAN Message Buffer index.                     |
| <i>pRxFrame</i> | Pointer to CAN message frame structure for reception. |

## Return values

## FlexCAN Driver

|                                    |                                                                           |
|------------------------------------|---------------------------------------------------------------------------|
| <i>kStatus_Success</i>             | - Rx Message Buffer is full and has been read successfully.               |
| <i>kStatus_FLEXCAN_Rx-Overflow</i> | - Rx Message Buffer is already overflowed and has been read successfully. |
| <i>kStatus_Fail</i>                | - Rx Message Buffer is empty.                                             |

### 20.2.7.27 **status\_t FLEXCAN\_ReadRxFifo ( CAN\_Type \* *base*, flexcan\_frame\_t \* *pRxFrame* )**

This function reads a CAN message from the FlexCAN build-in Rx FIFO.

Parameters

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.                      |
| <i>pRxFrame</i> | Pointer to CAN message frame structure for reception. |

Return values

|                        |                                           |
|------------------------|-------------------------------------------|
| <i>kStatus_Success</i> | - Read Message from Rx FIFO successfully. |
| <i>kStatus_Fail</i>    | - Rx FIFO is not enabled.                 |

### 20.2.7.28 **status\_t FLEXCAN\_TransferSendBlocking ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_frame\_t \* *pTxFrame* )**

Note that a transfer handle does not need to be created before calling this API.

Parameters

|                 |                                          |
|-----------------|------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base pointer.         |
| <i>mbIdx</i>    | The FlexCAN Message Buffer index.        |
| <i>pTxFrame</i> | Pointer to CAN message frame to be sent. |

Return values

|                        |                                          |
|------------------------|------------------------------------------|
| <i>kStatus_Success</i> | - Write Tx Message Buffer Successfully.  |
| <i>kStatus_Fail</i>    | - Tx Message Buffer is currently in use. |

**20.2.7.29** `status_t FLEXCAN_TransferReceiveBlocking ( CAN_Type * base, uint8_t mbIdx, flexcan_frame_t * pRxFrame )`

Note that a transfer handle does not need to be created before calling this API.

## FlexCAN Driver

### Parameters

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base pointer.                      |
| <i>mbIdx</i>    | The FlexCAN Message Buffer index.                     |
| <i>pRxFrame</i> | Pointer to CAN message frame structure for reception. |

### Return values

|                                    |                                                                           |
|------------------------------------|---------------------------------------------------------------------------|
| <i>kStatus_Success</i>             | - Rx Message Buffer is full and has been read successfully.               |
| <i>kStatus_FLEXCAN_Rx-Overflow</i> | - Rx Message Buffer is already overflowed and has been read successfully. |
| <i>kStatus_Fail</i>                | - Rx Message Buffer is empty.                                             |

### 20.2.7.30 **status\_t FLEXCAN\_TransferReceiveFifoBlocking ( CAN\_Type \* *base*, flexcan\_frame\_t \* *pRxFrame* )**

Note that a transfer handle does not need to be created before calling this API.

### Parameters

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base pointer.                      |
| <i>pRxFrame</i> | Pointer to CAN message frame structure for reception. |

### Return values

|                        |                                           |
|------------------------|-------------------------------------------|
| <i>kStatus_Success</i> | - Read Message from Rx FIFO successfully. |
| <i>kStatus_Fail</i>    | - Rx FIFO is not enabled.                 |

### 20.2.7.31 **void FLEXCAN\_TransferCreateHandle ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_transfer\_callback\_t *callback*, void \* *userData* )**

This function initializes the FlexCAN handle, which can be used for other FlexCAN transactional APIs. Usually, for a specified FlexCAN instance, call this API once to get the initialized handle.

### Parameters

---



|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.        |
| <i>handle</i>   | FlexCAN handle pointer.                 |
| <i>callback</i> | The callback function.                  |
| <i>userData</i> | The parameter of the callback function. |

#### 20.2.7.32 **status\_t FLEXCAN\_TransferSendNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_mb\_transfer\_t \* *pMbXfer* )**

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

|                |                                                                                            |
|----------------|--------------------------------------------------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.                                                           |
| <i>handle</i>  | FlexCAN handle pointer.                                                                    |
| <i>pMbXfer</i> | FlexCAN Message Buffer transfer structure. See the <a href="#">flexcan_mb_transfer_t</a> . |

Return values

|                                |                                                       |
|--------------------------------|-------------------------------------------------------|
| <i>kStatus_Success</i>         | Start Tx Message Buffer sending process successfully. |
| <i>kStatus_Fail</i>            | Write Tx Message Buffer failed.                       |
| <i>kStatus_FLEXCAN_Tx-Busy</i> | Tx Message Buffer is in use.                          |

#### 20.2.7.33 **status\_t FLEXCAN\_TransferReceiveNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_mb\_transfer\_t \* *pMbXfer* )**

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

|                |                                                                                            |
|----------------|--------------------------------------------------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.                                                           |
| <i>handle</i>  | FlexCAN handle pointer.                                                                    |
| <i>pMbXfer</i> | FlexCAN Message Buffer transfer structure. See the <a href="#">flexcan_mb_transfer_t</a> . |

## FlexCAN Driver

### Return values

|                                |                                                           |
|--------------------------------|-----------------------------------------------------------|
| <i>kStatus_Success</i>         | - Start Rx Message Buffer receiving process successfully. |
| <i>kStatus_FLEXCAN_Rx-Busy</i> | - Rx Message Buffer is in use.                            |

### 20.2.7.34 **status\_t FLEXCAN\_TransferReceiveFifoNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_fifo\_transfer\_t \* *pFifoXfer* )**

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

### Parameters

|                  |                                                                                       |
|------------------|---------------------------------------------------------------------------------------|
| <i>base</i>      | FlexCAN peripheral base address.                                                      |
| <i>handle</i>    | FlexCAN handle pointer.                                                               |
| <i>pFifoXfer</i> | FlexCAN Rx FIFO transfer structure. See the <a href="#">flexcan_fifo_transfer_t</a> . |

### Return values

|                                    |                                                 |
|------------------------------------|-------------------------------------------------|
| <i>kStatus_Success</i>             | - Start Rx FIFO receiving process successfully. |
| <i>kStatus_FLEXCAN_Rx-FifoBusy</i> | - Rx FIFO is currently in use.                  |

### 20.2.7.35 **uint32\_t FLEXCAN\_GetTimeStamp ( flexcan\_handle\_t \* *handle*, uint8\_t *mbIdx* )**

Then function can only be used when calling non-blocking Data transfer (TX/RX) API, After TX/RX data transfer done (User can get the status by handler's callback function), we can get the detail index of Mailbox's timestamp by handle, Detail non-blocking data transfer API (TX/RX) contain. -FLEXCAN\_TransferSendNonBlocking -FLEXCAN\_TransferFDSendNonBlocking -FLEXCAN\_TransferReceiveNonBlocking -FLEXCAN\_TransferFDReceiveNonBlocking -FLEXCAN\_TransferReceiveFifoNonBlocking

### Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>handle</i> | FlexCAN handle pointer.              |
| <i>mbIdx</i>  | The FlexCAN FD Message Buffer index. |

Return values

|            |                                                     |
|------------|-----------------------------------------------------|
| <i>the</i> | index of mailbox 's timestamp stored in the handle. |
|------------|-----------------------------------------------------|

#### 20.2.7.36 void FLEXCAN\_TransferAbortSend ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, uint8\_t *mbIdx* )

This function aborts the interrupt driven message send process.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.  |
| <i>handle</i> | FlexCAN handle pointer.           |
| <i>mbIdx</i>  | The FlexCAN Message Buffer index. |

#### 20.2.7.37 void FLEXCAN\_TransferAbortReceive ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, uint8\_t *mbIdx* )

This function aborts the interrupt driven message receive process.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.  |
| <i>handle</i> | FlexCAN handle pointer.           |
| <i>mbIdx</i>  | The FlexCAN Message Buffer index. |

#### 20.2.7.38 void FLEXCAN\_TransferAbortReceiveFifo ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle* )

This function aborts the interrupt driven message receive from Rx FIFO process.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address. |
| <i>handle</i> | FlexCAN handle pointer.          |

### 20.2.7.39 void FLEXCAN\_TransferHandleIRQ ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle* )

This function handles the FlexCAN Error, the Message Buffer, and the Rx FIFO IRQ request.

## Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address. |
| <i>handle</i> | FlexCAN handle pointer.          |



## Chapter 21

### FTM: FlexTimer Driver

#### Overview

The MCUXpresso SDK provides a driver for the FlexTimer Module (FTM) of MCUXpresso SDK devices.

#### 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.

##### 21.2.1 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.

##### 21.2.2 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.

##### 21.2.3 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

## Register Update

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.

### 21.2.4 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.

### 21.2.5 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.

### 21.2.6 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.

## 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/ftm` Multiple 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/ftm`



## Typical use case

### 21.4.1 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\\_CombinedPwm](#),  
  [kFTM\\_ComplementaryPwm](#) }  
*FTM PWM operation modes.*

## Typical use case

- 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\_SHIFT)),  
    `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` }  
    *FlexTimer quadrature decode modes.*
- 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` {

## Typical use case

```
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\_SetupFault** (FTM\_Type \*base, **ftm\_fault\_input\_t** faultNumber, const **ftm\_fault\_param\_t** \*faultParams)  
*Sets up the working of the FTM fault 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, 2, 3))  
*FTM driver version 2.2.3.*

## Initialization and deinitialization

- **status\_t** **FTM\_Init** (FTM\_Type \*base, const **ftm\_config\_t** \*config)  
*Un-gates 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.*

## 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.*
- void **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)

## Typical use case

*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.*

## 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.*

## Data Structure Documentation

### 21.5.1 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 combined PWM mode to generate an asymmetrical PWM.*
- bool [enableDeadtime](#)  
*true: The deadtime insertion in this pair of channels is enabled; false: The deadtime insertion in this pair of channels is disabled.*

#### 21.5.1.0.0.9 Field Documentation

##### 21.5.1.0.0.9.1 [ftm\\_chnl\\_t](#) [ftm\\_chnl\\_pwm\\_signal\\_param\\_t::chnlNumber](#)

In combined mode, this represents the channel pair number.

##### 21.5.1.0.0.9.2 [ftm\\_pwm\\_level\\_select\\_t](#) [ftm\\_chnl\\_pwm\\_signal\\_param\\_t::level](#)

##### 21.5.1.0.0.9.3 [uint8\\_t](#) [ftm\\_chnl\\_pwm\\_signal\\_param\\_t::dutyCyclePercent](#)

100 = always active signal (100% duty cycle).

##### 21.5.1.0.0.9.4 [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

## Data Structure Documentation

21.5.1.0.0.9.5 `bool` `ftm_chnl_pwm_signal_param_t::enableDeadtime`

### 21.5.2 `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 combined PWM mode to generate an asymmetrical PWM.*

#### 21.5.2.0.0.10 Field Documentation

21.5.2.0.0.10.1 `ftm_chnl_t` `ftm_chnl_pwm_config_param_t::chnlNumber`

In combined mode, this represents the channel pair number.

21.5.2.0.0.10.2 `ftm_pwm_level_select_t` `ftm_chnl_pwm_config_param_t::level`

21.5.2.0.0.10.3 `uint16_t` `ftm_chnl_pwm_config_param_t::dutyValue`

21.5.2.0.0.10.4 `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.

### 21.5.3 `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.*

### 21.5.4 `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.*

### 21.5.5 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.*

### 21.5.6 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`

## Enumeration Type Documentation

- *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.*

### 21.5.6.0.0.11 Field Documentation

#### 21.5.6.0.0.11.1 uint32\_t ftm\_config\_t::pwmSyncMode

#### 21.5.6.0.0.11.2 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](#).

#### 21.5.6.0.0.11.3 uint32\_t ftm\_config\_t::deadTimeValue

#### 21.5.6.0.0.11.4 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](#).

## Macro Definition Documentation

### 21.6.1 #define FSL\_FTM\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 3))

## Enumeration Type Documentation

### 21.7.1 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.

### 21.7.2 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.

### 21.7.3 enum ftm\_pwm\_mode\_t

Enumerator

*kFTM\_EdgeAlignedPwm* Edge-aligned PWM.  
*kFTM\_CenterAlignedPwm* Center-aligned PWM.  
*kFTM\_CombinedPwm* Combined PWM.  
*kFTM\_ComplementaryPwm* Complementary PWM.

### 21.7.4 enum ftm\_pwm\_level\_select\_t

Enumerator

*kFTM\_NoPwmSignal* No PWM output on pin.  
*kFTM\_LowTrue* Low true pulses.  
*kFTM\_HighTrue* High true pulses.

### 21.7.5 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.

### 21.7.6 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.

## Enumeration Type Documentation

### 21.7.7 enum ftm\_dual\_edge\_capture\_mode\_t

Enumerator

*kFTM\_OneShot* One-shot capture mode.  
*kFTM\_Continuous* Continuous capture mode.

### 21.7.8 enum ftm\_quad\_decode\_mode\_t

Enumerator

*kFTM\_QuadPhaseEncode* Phase A and Phase B encoding mode.  
*kFTM\_QuadCountAndDir* Count and direction encoding mode.

### 21.7.9 enum ftm\_phase\_polarity\_t

Enumerator

*kFTM\_QuadPhaseNormal* Phase input signal is not inverted.  
*kFTM\_QuadPhaseInvert* Phase input signal is inverted.

### 21.7.10 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.

### 21.7.11 enum ftm\_clock\_source\_t

Enumerator

*kFTM\_SystemClock* System clock selected.  
*kFTM\_FixedClock* Fixed frequency clock.  
*kFTM\_ExternalClock* External clock.

### 21.7.12 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.

### 21.7.13 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.

### 21.7.14 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.

### 21.7.15 enum ftm\_external\_trigger\_t

## Enumeration Type Documentation

### 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.

## 21.7.16 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.

## 21.7.17 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.

### 21.7.18 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.

### 21.7.19 enum ftm\_status\_flags\_t

Note

Actual available flags are SoC-specific

Enumerator

***kFTM\_Chnl0Flag*** Channel 0 Flag.  
***kFTM\_Chnl1Flag*** Channel 1 Flag.  
***kFTM\_Chnl2Flag*** Channel 2 Flag.  
***kFTM\_Chnl3Flag*** Channel 3 Flag.  
***kFTM\_Chnl4Flag*** Channel 4 Flag.  
***kFTM\_Chnl5Flag*** Channel 5 Flag.  
***kFTM\_Chnl6Flag*** Channel 6 Flag.  
***kFTM\_Chnl7Flag*** Channel 7 Flag.  
***kFTM\_FaultFlag*** Fault Flag.  
***kFTM\_TimeOverflowFlag*** Time overflow Flag.  
***kFTM\_ChnlTriggerFlag*** Channel trigger Flag.  
***kFTM\_ReloadFlag*** Reload Flag; Available only on certain SoC's.

## Function Documentation

### 21.7.20 anonymous enum

Enumerator

***kFTM\_QuadDecoderCountingIncreaseFlag*** Counting direction is increasing (FTM counter increment), or the direction is decreasing.

***kFTM\_QuadDecoderCountingOverflowOnTopFlag*** Indicates if the TOF bit was set on the top or the bottom of counting.

## Function Documentation

### 21.8.1 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.

### 21.8.2 void FTM\_Deinit ( FTM\_Type \* *base* )

Parameters

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

### 21.8.3 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. |
|---------------|----------------------------------------------|

#### 21.8.4 **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

#### 21.8.5 **void FTM\_UpdatePwmDutycycle ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlNumber*, ftm\_pwm\_mode\_t *currentPwmMode*, uint8\_t *dutyCyclePercent* )**

## Parameters

## Function Documentation

|                          |                                                                                                                                   |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <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>currentPwm-Mode</i>   | The current PWM mode set during PWM setup                                                                                         |
| <i>dutyCycle-Percent</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) |

### 21.8.6 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. |

### 21.8.7 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

### 21.8.8 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. |

### 21.8.9 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.                            |

### 21.8.10 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.

## Function Documentation

### 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. |

### 21.8.11 void FTM\_SetupFault ( 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 a 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 |

### 21.8.12 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> |

### 21.8.13 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> |
|-------------|---------------------------------------------------------------------------------------------------------------------|

#### 21.8.14 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](#)

#### 21.8.15 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](#)

#### 21.8.16 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> |

#### 21.8.17 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.

## Function Documentation

### 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. |

#### 21.8.18 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

#### 21.8.19 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. |

#### 21.8.20 static void FTM\_StopTimer ( FTM\_Type \* *base* ) [inline], [static]

## Parameters

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

**21.8.21** `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 |

**21.8.22** `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 |

**21.8.23** `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 |

**21.8.24** `static void FTM_SetOutputMask ( FTM_Type * base, ftm_chnl_t chnlNumber, bool mask ) [inline], [static]`

## Function Documentation

### 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 |

**21.8.25** **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-<br/>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 |

**21.8.26** **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-<br/>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 |

**21.8.27** **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-<br/>Number</i> | The FTM channel pair number; options are 0, 1, 2, 3                |
| <i>value</i>                | true: enable complementary mode; false: disable complementary mode |



**21.8.28** `static void FTM_SetInvertEnable ( FTM_Type * base, ftm_chnl_t  
chnlPairNumber, bool value ) [inline], [static]`

## Function Documentation

### 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    |

**21.8.29 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 |

**21.8.30 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`.

**21.8.31 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. |

**21.8.32** `static uint32_t FTM_GetQuadDecoderCounterValue ( FTM_Type * base )  
[inline], [static]`

## Parameters

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

## Returns

Current quad Decoder counter value.

**21.8.33** `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. |
|-------------|------------------------------|

**21.8.34** `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 |

**21.8.35** `static void FTM_SetWriteProtection ( FTM_Type * base, bool enable )  
[inline], [static]`

## Function Documentation

### Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | FTM peripheral base address                                            |
| <i>enable</i> | true: Write-protection is enabled, false: Write-protection is disabled |

### 21.8.36 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 22

# GPIO: General-Purpose Input/Output Driver

### 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, 5, 1))  
*GPIO driver version 2.5.1.*

## Data Structure Documentation

### 22.2.1 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.*

## Enumeration Type Documentation

## Macro Definition Documentation

### 22.3.1 #define FSL\_GPIO\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 1))

## Enumeration Type Documentation

### 22.4.1 enum gpio\_pin\_direction\_t

Enumerator

***kGPIO\_DigitalInput*** Set current pin as digital input.

***kGPIO\_DigitalOutput*** Set current pin as digital output.

## GPIO Driver

### 22.5.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General-Purpose Input/Output (GPIO) module of MCUXpresso SDK devices.

### 22.5.2 Typical use case

#### 22.5.2.1 Output Operation

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

#### 22.5.2.2 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.*

## GPIO Driver

- void [GPIO\\_PortClearInterruptFlags](#) (GPIO\_Type \*base, uint32\_t mask)  
*Clears multiple GPIO pin interrupt status flags.*

### 22.5.3 Function Documentation

#### 22.5.3.1 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                                 |

#### 22.5.3.2 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> |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 22.5.3.3 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                                          |

### 22.5.3.4 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                                          |

### 22.5.3.5 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                                          |

### 22.5.3.6 static uint32\_t GPIO\_PinRead ( GPIO\_Type \* *base*, uint32\_t *pin* ) [inline], [static]

Parameters

## GPIO Driver

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <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> |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 22.5.3.7 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. |
|------------|-------------------------------------------------------------------------------------------------------------|

### 22.5.3.8 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                                          |

## 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.

## 22.6.1 Typical use case

### 22.6.1.1 Output Operation

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

### 22.6.1.2 Input Operation

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





## Chapter 23

### I2C: Inter-Integrated Circuit Driver

#### Overview

#### Modules

- [I2C CMSIS Driver](#)
- [I2C Driver](#)
- [I2C FreeRTOS Driver](#)
- [I2C eDMA Driver](#)

## I2C Driver

### 23.2.1 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.

### 23.2.2 Typical use case

#### 23.2.2.1 Master Operation in functional method

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/i2c`

#### 23.2.2.2 Master Operation in interrupt transactional method

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/i2c`

#### 23.2.2.3 Master Operation in DMA transactional method

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/i2c`

#### 23.2.2.4 Slave Operation in functional method

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/i2c`

#### 23.2.2.5 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 asserted/deassert. \*/  
*Retry times for waiting flag.*
- #define [I2C\\_MASTER\\_FACK\\_CONTROL](#) 0U /\* Default defines to zero means master will send ack automatically. \*/  
*Master 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.*

## I2C Driver

- 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 }  
    *I2C feature interrupt source.*
- 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\_SlaveGeneralCallEvent* = 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, 8))  
*I2C driver version 2.0.8.*

## 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 **I2C\_GetInstance** (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.*

## I2C Driver

### 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.*

### 23.2.3 Data Structure Documentation

#### 23.2.3.1 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.*

##### 23.2.3.1.0.12 Field Documentation

23.2.3.1.0.12.1 bool i2c\_master\_config\_t::enableMaster

23.2.3.1.0.12.2 bool i2c\_master\_config\_t::enableStopHold

23.2.3.1.0.12.3 uint32\_t i2c\_master\_config\_t::baudRate\_Bps

23.2.3.1.0.12.4 uint8\_t i2c\_master\_config\_t::glitchFilterWidth

#### 23.2.3.2 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.*

## I2C Driver

- `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.*

### 23.2.3.2.0.13 Field Documentation

**23.2.3.2.0.13.1** `bool i2c_slave_config_t::enableSlave`

**23.2.3.2.0.13.2** `bool i2c_slave_config_t::enableGeneralCall`

**23.2.3.2.0.13.3** `bool i2c_slave_config_t::enableWakeUp`

**23.2.3.2.0.13.4** `bool i2c_slave_config_t::enableBaudRateCtl`

**23.2.3.2.0.13.5** `uint16_t i2c_slave_config_t::slaveAddress`

**23.2.3.2.0.13.6** `uint16_t i2c_slave_config_t::upperAddress`

**23.2.3.2.0.13.7** `i2c_slave_address_mode_t i2c_slave_config_t::addressingMode`

**23.2.3.2.0.13.8** `uint32_t i2c_slave_config_t::sclStopHoldTime_ns`

### 23.2.3.3 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.*

**23.2.3.3.0.14 Field Documentation****23.2.3.3.0.14.1** `uint32_t i2c_master_transfer_t::flags`**23.2.3.3.0.14.2** `uint8_t i2c_master_transfer_t::slaveAddress`**23.2.3.3.0.14.3** `i2c_direction_t i2c_master_transfer_t::direction`**23.2.3.3.0.14.4** `uint32_t i2c_master_transfer_t::subaddress`

Transferred MSB first.

**23.2.3.3.0.14.5** `uint8_t i2c_master_transfer_t::subaddressSize`**23.2.3.3.0.14.6** `uint8_t* volatile i2c_master_transfer_t::data`**23.2.3.3.0.14.7** `volatile size_t i2c_master_transfer_t::dataSize`**23.2.3.4 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.*

## I2C Driver

### 23.2.3.4.0.15 Field Documentation

23.2.3.4.0.15.1 `i2c_master_transfer_t i2c_master_handle_t::transfer`

23.2.3.4.0.15.2 `size_t i2c_master_handle_t::transferSize`

23.2.3.4.0.15.3 `uint8_t i2c_master_handle_t::state`

23.2.3.4.0.15.4 `i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback`

23.2.3.4.0.15.5 `void* i2c_master_handle_t::userData`

### 23.2.3.5 struct `i2c_slave_transfer_t`

#### Data Fields

- `i2c_slave_transfer_event_t event`  
*A reason that the callback is invoked.*
- `uint8_t *volatile data`  
*A transfer buffer.*
- `volatile size_t dataSize`  
*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.*

### 23.2.3.5.0.16 Field Documentation

23.2.3.5.0.16.1 `i2c_slave_transfer_event_t i2c_slave_transfer_t::event`

23.2.3.5.0.16.2 `uint8_t* volatile i2c_slave_transfer_t::data`

23.2.3.5.0.16.3 `volatile size_t i2c_slave_transfer_t::dataSize`

23.2.3.5.0.16.4 `status_t i2c_slave_transfer_t::completionStatus`

Only applies for `kI2C_SlaveCompletionEvent`.

23.2.3.5.0.16.5 `size_t i2c_slave_transfer_t::transferredCount`

### 23.2.3.6 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.*

### 23.2.3.6.0.17 Field Documentation

23.2.3.6.0.17.1 `volatile bool i2c_slave_handle_t::isBusy`

23.2.3.6.0.17.2 `i2c_slave_transfer_t i2c_slave_handle_t::transfer`

23.2.3.6.0.17.3 `uint32_t i2c_slave_handle_t::eventMask`

23.2.3.6.0.17.4 `i2c_slave_transfer_callback_t i2c_slave_handle_t::callback`

23.2.3.6.0.17.5 `void* i2c_slave_handle_t::userData`

## 23.2.4 Macro Definition Documentation

23.2.4.1 `#define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 0, 8))`

23.2.4.2 `#define I2C_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`

## 23.2.5 Typedef Documentation

23.2.5.1 `typedef void(* i2c_master_transfer_callback_t)(I2C_Type *base, i2c_master_handle_t *handle, status_t status, void *userData)`

23.2.5.2 `typedef void(* i2c_slave_transfer_callback_t)(I2C_Type *base, i2c_slave_transfer_t *xfer, void *userData)`

## 23.2.6 Enumeration Type Documentation

### 23.2.6.1 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.

## I2C Driver

### 23.2.6.2 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.

### 23.2.6.3 enum \_i2c\_interrupt\_enable

Enumerator

*kI2C\_GlobalInterruptEnable* I2C global interrupt.  
*kI2C\_StartStopDetectInterruptEnable* I2C start&stop detect interrupt.

### 23.2.6.4 enum i2c\_direction\_t

Enumerator

*kI2C\_Write* Master transmits to the slave.  
*kI2C\_Read* Master receives from the slave.

### 23.2.6.5 enum i2c\_slave\_address\_mode\_t

Enumerator

*kI2C\_Address7bit* 7-bit addressing mode.  
*kI2C\_RangeMatch* Range address match addressing mode.



### 23.2.6.6 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.

### 23.2.6.7 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.

### 23.2.6.8 anonymous enum

Enumerator

***kClearFlags*** All flags which are cleared by the driver upon starting a transfer.

## 23.2.7 Function Documentation

### 23.2.7.1 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.

## I2C Driver

### 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            |

### 23.2.7.2 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           |

**23.2.7.3 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 |
|-------------|------------------|

**23.2.7.4 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 |
|-------------|------------------|

**23.2.7.5 uint32\_t I2C\_GetInstance ( I2C\_Type \* *base* )**

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | I2C peripheral base address. |
|-------------|------------------------------|

**23.2.7.6 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);
*
```

## I2C Driver

### Parameters

|                     |                                                  |
|---------------------|--------------------------------------------------|
| <i>masterConfig</i> | A pointer to the master configuration structure. |
|---------------------|--------------------------------------------------|

### 23.2.7.7 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. |
|--------------------|-------------------------------------------------|

### 23.2.7.8 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. |

### 23.2.7.9 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.

### 23.2.7.10 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.

### 23.2.7.11 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> | The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values: <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> |

### 23.2.7.12 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> | The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values: <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> |

### 23.2.7.13 void I2C\_EnableInterrupts ( I2C\_Type \* *base*, uint32\_t *mask* )

## I2C Driver

### 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> |

#### 23.2.7.14 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> |

#### 23.2.7.15 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 |

#### 23.2.7.16 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

**23.2.7.17 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               |

**23.2.7.18 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.                |

**23.2.7.19 status\_t I2C\_MasterStop ( I2C\_Type \* *base* )**

## I2C Driver

Return values

|                            |                                    |
|----------------------------|------------------------------------|
| <i>kStatus_Success</i>     | Successfully send the stop signal. |
| <i>kStatus_I2C_Timeout</i> | Send stop signal failed, timeout.  |

### 23.2.7.20 **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. |

### 23.2.7.21 **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 kI2C_Transfer-DefaultFlag to issue a stop and kI2C_TransferNoStop 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. |
|------------------------|----------------------------------------------|

### 23.2.7.22 **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 kI2C_TransferDefaultFlag to issue a stop and kI2C_TransferNoStop 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.            |

### 23.2.7.23 **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

## I2C Driver

|                                     |                                              |
|-------------------------------------|----------------------------------------------|
| <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. |

### 23.2.7.24 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.           |

### 23.2.7.25 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. |

#### 23.2.7.26 void I2C\_MasterTransferCreateHandle ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle*, i2c\_master\_transfer\_callback\_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.                       |

#### 23.2.7.27 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 <a href="#">i2c_master_transfer_t</a> structure.               |

Return values

|                        |                                           |
|------------------------|-------------------------------------------|
| <i>kStatus_Success</i> | Successfully start the data transmission. |
|------------------------|-------------------------------------------|

## I2C Driver

|                            |                                           |
|----------------------------|-------------------------------------------|
| <i>kStatus_I2C_Busy</i>    | Previous transmission still not finished. |
| <i>kStatus_I2C_Timeout</i> | Transfer error, wait signal timeout.      |

**23.2.7.28** `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> | count is Invalid.              |
| <i>kStatus_Success</i>         | Successfully return the count. |

**23.2.7.29** `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. |

**23.2.7.30** `void I2C_MasterTransferHandleIRQ ( I2C_Type * base, void * i2cHandle )`

## Parameters

|                  |                                           |
|------------------|-------------------------------------------|
| <i>base</i>      | I2C base pointer.                         |
| <i>i2cHandle</i> | pointer to i2c_master_handle_t structure. |

**23.2.7.31 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 i2c_slave_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.                      |

**23.2.7.32 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 kI2C\_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. |

## I2C Driver

### 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. |

### 23.2.7.33 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. |

### 23.2.7.34 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. |

### 23.2.7.35 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 |

## I2C eDMA Driver

## I2C eDMA Driver

### 23.3.1 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, 8))  
*I2C EDMA driver version 2.0.8.*

### 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.*

### 23.3.2 Data Structure Documentation

#### 23.3.2.1 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.*

## I2C eDMA Driver

### 23.3.2.1.0.18 Field Documentation

23.3.2.1.0.18.1 `i2c_master_transfer_t i2c_master_edma_handle_t::transfer`

23.3.2.1.0.18.2 `size_t i2c_master_edma_handle_t::transferSize`

23.3.2.1.0.18.3 `uint8_t i2c_master_edma_handle_t::nbytes`

23.3.2.1.0.18.4 `uint8_t i2c_master_edma_handle_t::state`

23.3.2.1.0.18.5 `edma_handle_t* i2c_master_edma_handle_t::dmaHandle`

23.3.2.1.0.18.6 `i2c_master_edma_transfer_callback_t i2c_master_edma_handle_t::completion-  
Callback`

23.3.2.1.0.18.7 `void* i2c_master_edma_handle_t::userData`

### 23.3.3 Macro Definition Documentation

23.3.3.1 `#define FSL_I2C_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 8))`

### 23.3.4 Typedef Documentation

23.3.4.1 `typedef void(* i2c_master_edma_transfer_callback_t)(I2C_Type *base,  
i2c_master_edma_handle_t *handle, status_t status, void *userData)`

### 23.3.5 Function Documentation

23.3.5.1 `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 <code>i2c_master_edma_handle_t</code> 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.                                              |

### 23.3.5.2 `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 <code>i2c_master_edma_handle_t</code> 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.   |

### 23.3.5.3 `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 <code>i2c_master_edma_handle_t</code> structure. |

## I2C eDMA Driver

|              |                                                                |
|--------------|----------------------------------------------------------------|
| <i>count</i> | A number of bytes transferred by the non-blocking transaction. |
|--------------|----------------------------------------------------------------|

**23.3.5.4 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. |

## I2C FreeRTOS Driver

### 23.4.1 Overview

#### Driver version

- #define `FSL_I2C_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 8)`)  
*I2C FreeRTOS driver version 2.0.8.*

### 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.*

### 23.4.2 Macro Definition Documentation

#### 23.4.2.1 #define FSL\_I2C\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 8))

### 23.4.3 Function Documentation

#### 23.4.3.1 `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

|                     |                                                                          |
|---------------------|--------------------------------------------------------------------------|
| <i>handle</i>       | The RTOS I2C handle, the pointer to an allocated space for RTOS context. |
| <i>base</i>         | The pointer base address of the I2C instance to initialize.              |
| <i>masterConfig</i> | The configuration structure to set-up I2C in master mode.                |
| <i>srcClock_Hz</i>  | The frequency of an input clock of the I2C module.                       |

Returns

status of the operation.

## I2C FreeRTOS Driver

### 23.4.3.2 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. |
|---------------|----------------------|

**23.4.3.3 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.

### 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.

#### 23.5.1 I2C CMSIS Driver

##### 23.5.1.1 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;
```

##### 23.5.1.2 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;

```

### 23.5.1.3 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 24

# LLWU: Low-Leakage Wakeup Unit Driver

### 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.

### 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.

### 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.

### 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.*

## Macro Definition Documentation

### Driver version

- #define **FSL\_LLWU\_DRIVER\_VERSION** (**MAKE\_VERSION**(2, 0, 5))  
*LLWU driver version.*

### Low-Leakage Wakeup Unit Control APIs

- void **LLWU\_SetExternalWakeupPinMode** (LLWU\_Type \*base, uint32\_t pinIndex, **llwu\_external\_pin\_mode\_t** pinMode)  
*Sets the external input pin source mode.*
- bool **LLWU\_GetExternalWakeupPinFlag** (LLWU\_Type \*base, uint32\_t pinIndex)  
*Gets the external wakeup source flag.*
- void **LLWU\_ClearExternalWakeupPinFlag** (LLWU\_Type \*base, uint32\_t pinIndex)  
*Clears the external wakeup source flag.*
- static void **LLWU\_EnableInternalModuleInterruptWakup** (LLWU\_Type \*base, uint32\_t moduleIndex, bool enable)  
*Enables/disables the internal module source.*
- static bool **LLWU\_GetInternalWakeupModuleFlag** (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.*
- void **LLWU\_SetResetPinMode** (LLWU\_Type \*base, bool pinEnable, bool pinFilterEnable)  
*Sets the reset pin mode.*
- #define **INTERNAL\_WAKEUP\_MODULE\_FLAG\_REG** F3

## Data Structure Documentation

### 24.5.1 struct llwu\_external\_pin\_filter\_mode\_t

#### Data Fields

- uint32\_t **pinIndex**  
*A pin number.*
- **llwu\_pin\_filter\_mode\_t** filterMode  
*Filter mode.*

## Macro Definition Documentation

### 24.6.1 #define FSL\_LLWU\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 5))

## Enumeration Type Documentation

### 24.7.1 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.

### 24.7.2 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.

## Function Documentation

### 24.8.1 void LLWU\_SetExternalWakeupPinMode ( 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.      |

### 24.8.2 bool LLWU\_GetExternalWakeupPinFlag ( 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.

## Function Documentation

### 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.

#### 24.8.3 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. |

#### 24.8.4 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                                             |

#### 24.8.5 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.

#### 24.8.6 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                                                    |

#### 24.8.7 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.

#### 24.8.8 void LLWU\_ClearPinFilterFlag ( LLWU\_Type \* *base*, uint32\_t *filterIndex* )

This function clears the pin filter flag.

## Function Documentation

### Parameters

|                    |                                                        |
|--------------------|--------------------------------------------------------|
| <i>base</i>        | LLWU peripheral base address.                          |
| <i>filterIndex</i> | A pin filter index to clear the flag, starting from 1. |

### 24.8.9 void LLWU\_SetResetPinMode ( LLWU\_Type \* *base*, bool *pinEnable*, bool *pinFilterEnable* )

This function determines how the reset pin is used as a low leakage mode exit source.

### Parameters

|                         |                                                                      |
|-------------------------|----------------------------------------------------------------------|
| <i>base</i>             | LLWU peripheral base address.                                        |
| <i>pinEnable</i>        | Enable reset the pin filter                                          |
| <i>pinFilter-Enable</i> | Specify whether the pin filter is enabled in Low-Leakage power mode. |



## Chapter 25

# LPTMR: Low-Power Timer

### Overview

The MCUXpresso SDK provides a driver for the Low-Power Timer (LPTMR) of MCUXpresso SDK devices.

### Function groups

The LPTMR driver supports operating the module as a time counter or as a pulse counter.

#### 25.2.1 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. 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.

#### 25.2.2 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.

#### 25.2.3 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.

## Typical use case

### 25.2.4 Status

Provides functions to get and clear the LPTMR status.

### 25.2.5 Interrupt

Provides functions to enable/disable LPTMR interrupts and get the currently enabled interrupts.

## Typical use case

### 25.3.1 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.*

## Data Structure Documentation

- 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.*

## Data Structure Documentation

### 25.4.1 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 clock source.*  
`lptmr_prescaler_glitch_value_t` value  
*Prescaler or glitch filter value.*

## Enumeration Type Documentation

### 25.5.1 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.

### 25.5.2 enum lptmr\_pin\_polarity\_t

Enumerator

- kLPTMR\_PinPolarityActiveHigh* Pulse Counter input source is active-high.
- kLPTMR\_PinPolarityActiveLow* Pulse Counter input source is active-low.

### 25.5.3 enum lptmr\_timer\_mode\_t

Enumerator

- kLPTMR\_TimerModeTimeCounter* Time Counter mode.
- kLPTMR\_TimerModePulseCounter* Pulse Counter mode.

### 25.5.4 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.

## Function Documentation

*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.

### 25.5.5 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.

### 25.5.6 enum lptmr\_interrupt\_enable\_t

Enumerator

*kLPTMR\_TimerInterruptEnable* Timer interrupt enable.

### 25.5.7 enum lptmr\_status\_flags\_t

Enumerator

*kLPTMR\_TimerCompareFlag* Timer compare flag.

## Function Documentation

### 25.6.1 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. |

**25.6.2 void LPTMR\_Deinit ( LPTMR\_Type \* *base* )**

## Parameters

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

**25.6.3 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. |
|---------------|-------------------------------------------------|

**25.6.4 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-<br/>_interrupt_enable_t</a> |

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

## Function Documentation

### 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> . |

### 25.6.6 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](#)

### 25.6.7 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](#)

### 25.6.8 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> . |

### 25.6.9 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. |

### 25.6.10 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

## Function Documentation

### 25.6.11 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 |
|-------------|-------------------------------|

### 25.6.12 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 26

# PDB: Programmable Delay Block

### 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 of 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.

### Typical use case

#### 26.2.1 Working as basic PDB counter with a PDB interrupt.

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

#### 26.2.2 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_PDBEIE_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` {

## Typical use case

```
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.*

## Data Structure Documentation

### 26.3.1 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.*

### 26.3.1.0.0.19 Field Documentation

26.3.1.0.0.19.1 `pdb_load_value_mode_t` `pdb_config_t::loadValueMode`

26.3.1.0.0.19.2 `pdb_prescaler_divider_t` `pdb_config_t::prescalerDivider`

26.3.1.0.0.19.3 `pdb_divider_multiplication_factor_t` `pdb_config_t::dividerMultiplicationFactor`

26.3.1.0.0.19.4 `pdb_trigger_input_source_t` `pdb_config_t::triggerInputSource`

26.3.1.0.0.19.5 `bool` `pdb_config_t::enableContinuousMode`

### 26.3.2 `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.*

### 26.3.2.0.0.20 Field Documentation

26.3.2.0.0.20.1 `uint32_t` `pdb_adc_pretrigger_config_t::enablePreTriggerMask`

26.3.2.0.0.20.2 `uint32_t` `pdb_adc_pretrigger_config_t::enableOutputMask`

PDB channel's corresponding pre-trigger asserts when the counter reaches the channel delay register.

26.3.2.0.0.20.3 `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.

### 26.3.3 `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.*



### 26.3.3.0.0.21 Field Documentation

26.3.3.0.0.21.1 `bool pdb_dac_trigger_config_t::enableExternalTriggerInput`

26.3.3.0.0.21.2 `bool pdb_dac_trigger_config_t::enableIntervalTrigger`

### Macro Definition Documentation

26.4.1 `#define FSL_PDB_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))`

### Enumeration Type Documentation

#### 26.5.1 `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.

#### 26.5.2 `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.

#### 26.5.3 `enum _pdb_interrupt_enable`

Enumerator

***kPDB\_SequenceErrorInterruptEnable*** PDB sequence error interrupt enable.

***kPDB\_DelayInterruptEnable*** PDB delay interrupt enable.

#### 26.5.4 `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)

## Enumeration Type Documentation

- 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.

### 26.5.5 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.

### 26.5.6 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.

### 26.5.7 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.  
***kPDB\_TriggerInput2*** Trigger-In 2.  
***kPDB\_TriggerInput3*** Trigger-In 3.  
***kPDB\_TriggerInput4*** Trigger-In 4.  
***kPDB\_TriggerInput5*** Trigger-In 5.  
***kPDB\_TriggerInput6*** Trigger-In 6.  
***kPDB\_TriggerInput7*** Trigger-In 7.  
***kPDB\_TriggerInput8*** Trigger-In 8.  
***kPDB\_TriggerInput9*** Trigger-In 9.  
***kPDB\_TriggerInput10*** Trigger-In 10.  
***kPDB\_TriggerInput11*** Trigger-In 11.  
***kPDB\_TriggerInput12*** Trigger-In 12.  
***kPDB\_TriggerInput13*** Trigger-In 13.  
***kPDB\_TriggerInput14*** Trigger-In 14.  
***kPDB\_TriggerSoftware*** Trigger-In 15, software trigger.

### 26.5.8 enum pdb\_adc\_trigger\_channel\_t

## Note

Actual number of available channels is SoC dependent

## Enumerator

***kPDB\_ADCTriggerChannel0*** PDB ADC trigger channel number 0.  
***kPDB\_ADCTriggerChannel1*** PDB ADC trigger channel number 1.  
***kPDB\_ADCTriggerChannel2*** PDB ADC trigger channel number 2.  
***kPDB\_ADCTriggerChannel3*** PDB ADC trigger channel number 3.

### 26.5.9 enum pdb\_adc\_pretrigger\_t

## Note

Actual number of available pretrigger channels is SoC dependent

## Enumerator

***kPDB\_ADCPreTrigger0*** PDB ADC pretrigger number 0.  
***kPDB\_ADCPreTrigger1*** PDB ADC pretrigger number 1.  
***kPDB\_ADCPreTrigger2*** PDB ADC pretrigger number 2.  
***kPDB\_ADCPreTrigger3*** PDB ADC pretrigger number 3.

## Enumeration Type Documentation

***kPDB\_ADCPreTrigger4*** PDB ADC pretrigger number 4.  
***kPDB\_ADCPreTrigger5*** PDB ADC pretrigger number 5.  
***kPDB\_ADCPreTrigger6*** PDB ADC pretrigger number 6.  
***kPDB\_ADCPreTrigger7*** PDB ADC pretrigger number 7.

### 26.5.10 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.

### 26.5.11 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.

### 26.5.12 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.

## Function Documentation

### 26.6.1 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". |

### 26.6.2 void PDB\_Deinit ( PDB\_Type \* *base* )

Parameters

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

### 26.6.3 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". |
|---------------|---------------------------------------------------------|

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

## Function Documentation

### Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PDB peripheral base address. |
| <i>enable</i> | Enable the module or not.    |

### 26.6.5 static void PDB\_DoSoftwareTrigger ( PDB\_Type \* *base* ) [inline], [static]

### Parameters

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

### 26.6.6 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. |
|-------------|------------------------------|

### 26.6.7 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.   |

### 26.6.8 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". |

### 26.6.9 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". |

### 26.6.10 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".

### 26.6.11 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". |

### 26.6.12 static void PDB\_SetModulusValue ( PDB\_Type \* *base*, uint32\_t *value* ) [inline], [static]

## Function Documentation

### Parameters

|              |                                                     |
|--------------|-----------------------------------------------------|
| <i>base</i>  | PDB peripheral base address.                        |
| <i>value</i> | Setting value for the modulus. 16-bit is available. |

**26.6.13** `static uint32_t PDB_GetCounterValue ( PDB_Type * base ) [inline], [static]`

### Parameters

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

### Returns

PDB counter's current value.

**26.6.14** `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. |

**26.6.15** `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". |
|---------------|----------------------------------------------------------------------------|

**26.6.16 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-<br/>Number</i> | Channel group index for ADC instance.                               |
| <i>value</i>                  | Setting value for ADC pre-trigger delay event. 16-bit is available. |

**26.6.17 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".

**26.6.18 static void PDB\_ClearADCPreTriggerStatusFlags ( PDB\_Type \* *base*, pdb\_adc\_trigger\_channel\_t *channel*, uint32\_t *mask* ) [inline], [static]**

## Function Documentation

### 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". |

**26.6.19** `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". |

**26.6.20** `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. |

**26.6.21** `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.                     |

**26.6.22 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 27

# PIT: Periodic Interrupt Timer

### Overview

The MCUXpresso SDK provides a driver for the Periodic Interrupt Timer (PIT) of MCUXpresso SDK devices.

### Function groups

The PIT driver supports operating the module as a time counter.

#### 27.2.1 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.

#### 27.2.2 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.

#### 27.2.3 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.

## Typical use case

### 27.2.4 Status

Provides functions to get and clear the PIT status.

### 27.2.5 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

## Typical use case

### 27.3.1 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, 2))  
*PIT Driver Version 2.0.2.*

## Initialization and deinitialization

- void `PIT_Init` (`PIT_Type` \*base, const `pit_config_t` \*config)  
*Un-gates 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.*

## Data Structure Documentation

### 27.4.1 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*

## Enumeration Type Documentation

### 27.5.1 enum pit\_chnl\_t

## Function Documentation

### 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.

## 27.5.2 enum pit\_interrupt\_enable\_t

### Enumerator

***kPIT\_TimerInterruptEnable*** Timer interrupt enable.

## 27.5.3 enum pit\_status\_flags\_t

### Enumerator

***kPIT\_TimerFlag*** Timer flag.

## Function Documentation

### 27.6.1 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 |

### 27.6.2 void PIT\_Deinit ( PIT\_Type \* *base* )



## Parameters

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

### 27.6.3 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. |
|---------------|-----------------------------------------|

### 27.6.4 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. |

### 27.6.5 static void PIT\_EnableInterrupts ( PIT\_Type \* *base*, pit\_chnl\_t *channel*, uint32\_t *mask* ) [inline], [static]

## Function Documentation

### 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> |

**27.6.6** `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> |

**27.6.7** `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](#)

**27.6.8** `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](#)

**27.6.9 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> |

**27.6.10 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        |

## Function Documentation

|              |                                |
|--------------|--------------------------------|
| <i>count</i> | Timer period in units of ticks |
|--------------|--------------------------------|

### 27.6.11 **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

### 27.6.12 **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.       |

### 27.6.13 **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 28

# PMC: Power Management Controller

### 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.*

## Data Structure Documentation

- 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.*

## Data Structure Documentation

### 28.2.1 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.*

### 28.2.2 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.*

### 28.2.3 struct pmc\_bandgap\_buffer\_config\_t

#### Data Fields

- bool [enable](#)  
*Enable bandgap buffer.*
- bool [enableInLowPowerMode](#)



*Enable bandgap buffer in low-power mode.*

### 28.2.3.0.0.22 Field Documentation

28.2.3.0.0.22.1 `bool pmc_bandgap_buffer_config_t::enable`

28.2.3.0.0.22.2 `bool pmc_bandgap_buffer_config_t::enableInLowPowerMode`

### Macro Definition Documentation

28.3.1 `#define FSL_PMC_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))`

Version 2.0.3.

### Enumeration Type Documentation

28.4.1 `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 )

28.4.2 `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)

### Function Documentation

28.5.1 `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

---

## Function Documentation

|               |                                             |
|---------------|---------------------------------------------|
| <i>base</i>   | PMC peripheral base address.                |
| <i>config</i> | Low-voltage detect configuration structure. |

### 28.5.2 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

### 28.5.3 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. |
|-------------|------------------------------|

### 28.5.4 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. |

### 28.5.5 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.

### 28.5.6 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. |
|-------------|------------------------------|

### 28.5.7 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

---

## Function Documentation

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

### 28.5.8 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.

### 28.5.9 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. |
|-------------|------------------------------|

### 28.5.10 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 29

# PORT: Port Control and Interrupts

### Overview

The MCUXpresso SDK provides a driver for the Port Control and Interrupts (PORT) module of MCU-Xpresso 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](#) {

## Overview

```
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, 1, 0))  
*Version 2.1.0.*

## Configuration

- static void `PORT_SetPinConfig` (`PORT_Type` \*base, `uint32_t` pin, const `port_pin_config_t` \*config)



- *Sets the port PCR register.*
- static void [PORT\\_SetMultiplePinsConfig](#) (PORT\_Type \*base, uint32\_t mask, const [port\\_pin\\_config\\_t](#) \*config)
- *Sets the port PCR register for multiple pins.*
- static void [PORT\\_SetPinMux](#) (PORT\_Type \*base, uint32\_t pin, [port\\_mux\\_t](#) mux)
- *Configures the pin muxing.*
- static void [PORT\\_EnablePinsDigitalFilter](#) (PORT\_Type \*base, uint32\_t mask, bool enable)
- *Enables the digital filter in one port, each bit of the 32-bit register represents one pin.*
- static void [PORT\\_SetDigitalFilterConfig](#) (PORT\_Type \*base, const [port\\_digital\\_filter\\_config\\_t](#) \*config)
- *Sets the digital filter in one port, each bit of the 32-bit register represents one pin.*

## 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.*

## Data Structure Documentation

### 29.2.1 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.*

### 29.2.2 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.*

## Enumeration Type Documentation

- uint16\_t [mux](#): 3  
*Pin mux Configure.*
- uint16\_t [lockRegister](#): 1  
*Lock/unlock the PCR field[15:0].*

## Macro Definition Documentation

### 29.3.1 #define FSL\_PORT\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 0))

## Enumeration Type Documentation

### 29.4.1 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.

### 29.4.2 enum \_port\_slew\_rate

Enumerator

***kPORT\_FastSlewRate*** Fast slew rate is configured.  
***kPORT\_SlowSlewRate*** Slow slew rate is configured.

### 29.4.3 enum \_port\_open\_drain\_enable

Enumerator

***kPORT\_OpenDrainDisable*** Open drain output is disabled.  
***kPORT\_OpenDrainEnable*** Open drain output is enabled.

### 29.4.4 enum \_port\_passive\_filter\_enable

Enumerator

***kPORT\_PassiveFilterDisable*** Passive input filter is disabled.  
***kPORT\_PassiveFilterEnable*** Passive input filter is enabled.

### 29.4.5 enum \_port\_drive\_strength

Enumerator

- kPORT\_LowDriveStrength* Low-drive strength is configured.
- kPORT\_HighDriveStrength* High-drive strength is configured.

### 29.4.6 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.

### 29.4.7 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.

### 29.4.8 enum port\_interrupt\_t

Enumerator

- kPORT\_InterruptOrDMADisabled* Interrupt/DMA request is disabled.
- kPORT\_DMARisingEdge* DMA request on rising edge.
- kPORT\_DMAFallingEdge* DMA request on falling edge.

## Function Documentation

***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.

### 29.4.9 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.

## Function Documentation

### 29.5.1 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. |

### 29.5.2 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. |

### 29.5.3 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> |

## Function Documentation

### 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

### 29.5.4 `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. |

### 29.5.5 `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. |

### 29.5.6 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> | PORT pin interrupt configuration. <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_DMAFallingEdge</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> |

### 29.5.7 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.              |

## Function Documentation

|                 |                                                                                                                                                                                                                                                       |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <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> |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 29.5.8 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.

### 29.5.9 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 30

# RCM: Reset Control Module Driver

### 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)

## Enumeration Type Documentation

- Gets the reset source status which caused a previous reset.*
- void [RCM\\_ConfigureResetPinFilter](#) (RCM\_Type \*base, const [rcm\\_reset\\_pin\\_filter\\_config\\_t](#) \*config)  
*Configures the reset pin filter.*
- static bool [RCM\\_GetEasyPortModePinStatus](#) (RCM\_Type \*base)  
*Gets the EZP\_MS\_B pin assert status.*

## Data Structure Documentation

### 30.2.1 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.*

#### 30.2.1.0.0.23 Field Documentation

30.2.1.0.0.23.1 bool rcm\_reset\_pin\_filter\_config\_t::enableFilterInStop

30.2.1.0.0.23.2 rcm\_run\_wait\_filter\_mode\_t rcm\_reset\_pin\_filter\_config\_t::filterInRunWait

30.2.1.0.0.23.3 uint8\_t rcm\_reset\_pin\_filter\_config\_t::busClockFilterCount

## Macro Definition Documentation

30.3.1 #define FSL\_RCM\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 4))

## Enumeration Type Documentation

### 30.4.1 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.

### 30.4.2 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.

## Function Documentation

### 30.5.1 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.

### 30.5.2 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.

## Function Documentation

### Parameters

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

### 30.5.3 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 31

# RNGA: Random Number Generator Accelerator Driver

### Overview

The MCUXpresso SDK provides a peripheral driver for the Random Number Generator Accelerator (RNGA) block of MCUXpresso SDK devices.

### 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.

### Get random data from RNGA

1. [RNGA\\_GetRandomData\(\)](#) function gets random data from the RNGA module.

### 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.

### 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.

## Macro Definition Documentation

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.*

## Macro Definition Documentation

### 31.6.1 #define FSL\_RNGA\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))

## Enumeration Type Documentation

### 31.7.1 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.

## Function Documentation

### 31.8.1 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 |
|-------------|-------------------|

### 31.8.2 void RNGA\_Deinit ( RNG\_Type \* *base* )

This function shuts down the RNGA.

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | RNGA base address |
|-------------|-------------------|

### 31.8.3 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 |

## Function Documentation

|                  |                       |
|------------------|-----------------------|
| <i>data_size</i> | size of data in bytes |
|------------------|-----------------------|

Returns

RNGA status

### 31.8.4 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  |

### 31.8.5 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 |

### 31.8.6 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 32

# RTC: Real Time Clock

### Overview

The MCUXpresso SDK provides a driver for the Real Time Clock (RTC) of MCUXpresso SDK devices.

### Function groups

The RTC driver supports operating the module as a time counter.

#### 32.2.1 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.

#### 32.2.2 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.

#### 32.2.3 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.

#### 32.2.4 Start & Stop timer

The function [RTC\\_StartTimer\(\)](#) starts the RTC time counter.

The function [RTC\\_StopTimer\(\)](#) stops the RTC time counter.

## Typical use case

### 32.2.5 Status

Provides functions to get and clear the RTC status.

### 32.2.6 Interrupt

Provides functions to enable/disable RTC interrupts and get current enabled interrupts.

### 32.2.7 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.

### 32.2.8 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.

## Typical use case

### 32.3.1 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)  
*Un gates 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.*

## Data Structure Documentation

### 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.*

## Data Structure Documentation

### 32.4.1 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.*

### 32.4.1.0.0.24 Field Documentation

32.4.1.0.0.24.1 `uint16_t rtc_datetime_t::year`

32.4.1.0.0.24.2 `uint8_t rtc_datetime_t::month`

32.4.1.0.0.24.3 `uint8_t rtc_datetime_t::day`

32.4.1.0.0.24.4 `uint8_t rtc_datetime_t::hour`

32.4.1.0.0.24.5 `uint8_t rtc_datetime_t::minute`

32.4.1.0.0.24.6 `uint8_t rtc_datetime_t::second`

### 32.4.2 `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.*

## Enumeration Type Documentation

### 32.5.1 `enum rtc_interrupt_enable_t`

Enumerator

***kRTC\_TimeInvalidInterruptEnable*** Time invalid interrupt.  
***kRTC\_TimeOverflowInterruptEnable*** Time overflow interrupt.  
***kRTC\_AlarmInterruptEnable*** Alarm interrupt.  
***kRTC\_SecondsInterruptEnable*** Seconds interrupt.

## Function Documentation

### 32.5.2 enum rtc\_status\_flags\_t

Enumerator

*kRTC\_TimeInvalidFlag* Time invalid flag.  
*kRTC\_TimeOverflowFlag* Time overflow flag.  
*kRTC\_AlarmFlag* Alarm flag.

### 32.5.3 enum rtc\_osc\_cap\_load\_t

Enumerator

*kRTC\_Capacitor\_2p* 2 pF capacitor load  
*kRTC\_Capacitor\_4p* 4 pF capacitor load  
*kRTC\_Capacitor\_8p* 8 pF capacitor load  
*kRTC\_Capacitor\_16p* 16 pF capacitor load

## Function Documentation

### 32.6.1 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. |

### 32.6.2 static void RTC\_Deinit ( RTC\_Type \* *base* ) [inline], [static]

Parameters

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

### 32.6.3 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. |
|---------------|----------------------------------------------------|

### 32.6.4 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

### 32.6.5 void RTC\_GetDatetime ( RTC\_Type \* base, rtc\_datetime\_t \* datetime )

## Function Documentation

### Parameters

|                 |                                                                      |
|-----------------|----------------------------------------------------------------------|
| <i>base</i>     | RTC peripheral base address                                          |
| <i>datetime</i> | Pointer to the structure where the date and time details are stored. |

### 32.6.6 **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

### 32.6.7 **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. |

### 32.6.8 **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 <a href="#">rtc_interrupt_enable_t</a> |
|-------------|---------------------------------------------------------------------------------------------------------------------|

### 32.6.9 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 <a href="#">rtc_interrupt_enable_t</a> |

### 32.6.10 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](#)

### 32.6.11 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](#)

### 32.6.12 void RTC\_ClearStatusFlags ( RTC\_Type \* *base*, uint32\_t *mask* )

## Function Documentation

### Parameters

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

### 32.6.13 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.

### 32.6.14 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 |
|-------------|-----------------------------|

### 32.6.15 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 |
|-------------|-----------------------------|

### 32.6.16 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 <a href="#">rtc_osc_cap_load_t</a> |

**32.6.17 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 |
|-------------|-----------------------------|

**32.6.18 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 33

# SAI: Serial Audio Interface

### 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.

### 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`

## Typical use case

## Typical use case

### 33.3.1 SAI Send/receive using an interrupt method

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

### 33.3.2 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](#)

## SAI Driver

### 33.4.1 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 }
  - Master 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 }

## SAI Driver

*The reset type.*

- 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, 1))  
*Version 2.3.1.*

## Initialization and deinitialization

- void `SAI_TxInit` (I2S\_Type \*base, const `sai_config_t` \*config)  
*Initializes the SAI Tx peripheral.*
- void `SAI_RxInit` (I2S\_Type \*base, const `sai_config_t` \*config)  
*Initializes the SAI Rx peripheral.*
- void `SAI_TxGetDefaultConfig` (`sai_config_t` \*config)  
*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 Driver

*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](#) type)

*Do software reset or FIFO reset.*

- void [SAI\\_RxSoftwareReset](#) ([I2S\\_Type](#) \*base, [sai\\_reset\\_type\\_t](#) type)

*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.*

## 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 uint32\_t [SAI\\_TxGetDataRegisterAddress](#) (I2S\_Type \*base, uint32\_t channel)  
*Gets the SAI Tx data register address.*
- static uint32\_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)  
*Terminate all SAI receive.*
- 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.*

### 33.4.2 Data Structure Documentation

#### 33.4.2.1 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.*

### 33.4.2.2 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](#)  
*Which audio protocol used.*
- [bool isFrameSyncCompact](#)  
*True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.*

#### 33.4.2.2.0.25 Field Documentation

##### 33.4.2.2.0.25.1 bool sai\_transfer\_format\_t::isFrameSyncCompact

### 33.4.2.3 struct sai\_master\_clock\_t

#### Data Fields

- [bool mclkOutputEnable](#)  
*master clock output enable*



## SAI Driver

- [sai\\_mclk\\_source\\_t mclkSource](#)  
*Master Clock source.*
- [uint32\\_t mclkHz](#)  
*target mclk frequency*
- [uint32\\_t mclkSourceClkHz](#)  
*mclk source frequency*

### 33.4.2.4 struct sai\_fifo\_t

#### Data Fields

- [uint8\\_t fifoWatermark](#)  
*fifo watermark*

### 33.4.2.5 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*

#### 33.4.2.5.0.26 Field Documentation

##### 33.4.2.5.0.26.1 bool sai\_bit\_clock\_t::bclkInputDelay

### 33.4.2.6 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.*
- [sai\\_clock\\_polarity\\_t frameSyncPolarity](#)  
*frame sync polarity*



### 33.4.2.7 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*

### 33.4.2.8 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.*

### 33.4.2.9 struct sai\_transfer\_t

#### Data Fields

- [uint8\\_t \\* data](#)

## SAI Driver

- *Data start address to transfer.*  
size\_t [dataSize](#)  
*Transfer size.*

### 33.4.2.9.0.27 Field Documentation

#### 33.4.2.9.0.27.1 uint8\_t\* sai\_transfer\_t::data

#### 33.4.2.9.0.27.2 size\_t sai\_transfer\_t::dataSize

### 33.4.2.10 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.*

### 33.4.3 Macro Definition Documentation

#### 33.4.3.1 #define SAI\_XFER\_QUEUE\_SIZE (4U)

### 33.4.4 Enumeration Type Documentation

#### 33.4.4.1 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.

#### 33.4.4.2 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

#### 33.4.4.3 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.

## SAI Driver

### 33.4.4.4 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

### 33.4.4.5 enum sai\_mono\_stereo\_t

Enumerator

*kSAI\_Stereo* Stereo sound.

*kSAI\_MonoRight* Only Right channel have sound.

*kSAI\_MonoLeft* Only left channel have sound.

### 33.4.4.6 enum sai\_data\_order\_t

Enumerator

*kSAI\_DataLSB* LSB bit transferred first.

*kSAI\_DataMSB* MSB bit transferred first.

### 33.4.4.7 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.

### 33.4.4.8 enum sai\_sync\_mode\_t

Enumerator

*kSAI\_ModeAsync* Asynchronous mode.

*kSAI\_ModeSync* Synchronous mode (with receiver or transmit)

**33.4.4.9 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.

**33.4.4.10 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.

**33.4.4.11 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\_FIFORequestInterruptEnable* FIFO request, means reached watermark.

**33.4.4.12 anonymous enum**

Enumerator

*kSAI\_FIFOWarningDMAEnable* FIFO warning caused by the DMA request.  
*kSAI\_FIFORequestDMAEnable* FIFO request caused by the DMA request.

**33.4.4.13 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.

## SAI Driver

*kSAI\_FIFORequestFlag* FIFO request flag.  
*kSAI\_FIFOWarningFlag* FIFO warning flag.

### 33.4.4.14 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.

### 33.4.4.15 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.

### 33.4.4.16 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.

### 33.4.4.17 enum sai\_transceiver\_type\_t

Enumerator

*kSAI\_Transmitter* sai transmitter  
*kSAI\_Receiver* sai receiver

### 33.4.4.18 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.

## 33.4.5 Function Documentation

### 33.4.5.1 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. |

### 33.4.5.2 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.

## SAI Driver

### Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SAI base pointer             |
| <i>config</i> | SAI configuration structure. |

#### 33.4.5.3 void SAI\_TxGetDefaultConfig ( sai\_config\_t \* *config* )

**Deprecated** Do not use this function. It has been superseded by [SAI\\_GetClassicI2SConfig](#), [SAI\\_GetLeftJustifiedConfig](#) , [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 |
|---------------|-------------------------------------------|

#### 33.4.5.4 void SAI\_RxGetDefaultConfig ( sai\_config\_t \* *config* )

**Deprecated** Do not use this function. It has been superseded by [SAI\\_GetClassicI2SConfig](#), [SAI\\_GetLeftJustifiedConfig](#) , [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 |
|---------------|-------------------------------------------|

#### 33.4.5.5 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. |
|-------------|-------------------|

### 33.4.5.6 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. |
|-------------|-------------------|

### 33.4.5.7 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 |
|-------------|------------------|

### 33.4.5.8 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 |
|-------------|------------------|

### 33.4.5.9 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. |

### 33.4.5.10 void SAI\_RxEnable ( I2S\_Type \* *base*, bool *enable* )

## SAI Driver

### Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | SAI base pointer.                              |
| <i>enable</i> | True means enable SAI Rx, false means disable. |

#### 33.4.5.11 **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. |

#### 33.4.5.12 **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. |

#### 33.4.5.13 **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. |

#### 33.4.5.14 **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. |

**33.4.5.15 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.      |

**33.4.5.16 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.      |

**33.4.5.17 void SAI\_TxSetBitclockConfig ( I2S\_Type \* *base*, sai\_master\_slave\_t *masterSlave*, sai\_bit\_clock\_t \* *config* )**

## SAI Driver

### 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. |

**33.4.5.18 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. |

**33.4.5.19 void SAI\_SetMasterClockConfig ( I2S\_Type \* *base*, sai\_master\_clock\_t \* *config* )**

### Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SAI base pointer.            |
| <i>config</i> | master clock configurations. |

**33.4.5.20 void SAI\_TxSetFifoConfig ( I2S\_Type \* *base*, sai\_fifo\_t \* *config* )**

### Parameters

|               |                      |
|---------------|----------------------|
| <i>base</i>   | SAI base pointer.    |
| <i>config</i> | fifo configurations. |

**33.4.5.21 void SAI\_RxSetFifoConfig ( I2S\_Type \* *base*, sai\_fifo\_t \* *config* )**

## Parameters

|               |                      |
|---------------|----------------------|
| <i>base</i>   | SAI base pointer.    |
| <i>config</i> | fifo configurations. |

**33.4.5.22 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. |

**33.4.5.23 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. |

**33.4.5.24 void SAI\_TxSetSerialDataConfig ( I2S\_Type \* *base*, sai\_serial\_data\_t \* *config* )**

## Parameters

|               |                             |
|---------------|-----------------------------|
| <i>base</i>   | SAI base pointer.           |
| <i>config</i> | serial data configurations. |

**33.4.5.25 void SAI\_RxSetSerialDataConfig ( I2S\_Type \* *base*, sai\_serial\_data\_t \* *config* )**

## SAI Driver

### Parameters

|               |                             |
|---------------|-----------------------------|
| <i>base</i>   | SAI base pointer.           |
| <i>config</i> | serial data configurations. |

### 33.4.5.26 void SAI\_TxSetConfig ( I2S\_Type \* *base*, sai\_transceiver\_t \* *config* )

### Parameters

|               |                             |
|---------------|-----------------------------|
| <i>base</i>   | SAI base pointer.           |
| <i>config</i> | transmitter configurations. |

### 33.4.5.27 void SAI\_RxSetConfig ( I2S\_Type \* *base*, sai\_transceiver\_t \* *config* )

### Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>config</i> | receiver configurations. |

### 33.4.5.28 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. |

### 33.4.5.29 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. |

**33.4.5.30** 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. |

**33.4.5.31** 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. |

**33.4.5.32** 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* )



## Parameters

|                        |                                      |
|------------------------|--------------------------------------|
| <i>config</i>          | transceiver configurations.          |
| <i>frameSync-Width</i> | length of frame sync.                |
| <i>bitWidth</i>        | audio data bitWidth.                 |
| <i>mode</i>            | audio data channel.                  |
| <i>saiChannel-Mask</i> | mask value of the channel to enable. |

### 33.4.5.33 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.

### 33.4.5.34 static void SAI\_TxClearStatusFlags ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                                                                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                           |
| <i>mask</i> | State mask. It can be a combination of the following source if defined: <ul style="list-style-type: none"> <li>• kSAI_WordStartFlag</li> <li>• kSAI_SyncErrorFlag</li> <li>• kSAI_FIFOErrorFlag</li> </ul> |

### 33.4.5.35 static uint32\_t SAI\_RxGetStatusFlag ( I2S\_Type \* *base* ) [inline], [static]

## SAI Driver

### Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | SAI base pointer |
|-------------|------------------|

### Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

#### 33.4.5.36 static void SAI\_RxClearStatusFlags ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

### Parameters

|             |                                                                                                                                                                                                         |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                        |
| <i>mask</i> | State mask. It can be a combination of the following sources if defined. <ul style="list-style-type: none"><li>• kSAI_WordStartFlag</li><li>• kSAI_SyncErrorFlag</li><li>• kSAI_FIFOErrorFlag</li></ul> |

#### 33.4.5.37 void SAI\_TxSoftwareReset ( I2S\_Type \* *base*, sai\_reset\_type\_t *type* )

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>type</i> | Reset type, FIFO reset or software reset |

#### 33.4.5.38 void SAI\_RxSoftwareReset ( I2S\_Type \* *base*, sai\_reset\_type\_t *type* )

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>type</i> | Reset type, FIFO reset or software reset |

### 33.4.5.39 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. |

### 33.4.5.40 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. |

### 33.4.5.41 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 |

### 33.4.5.42 void SAI\_RxSetDataOrder ( I2S\_Type \* *base*, sai\_data\_order\_t *order* )

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | SAI base pointer |
|-------------|------------------|

## SAI Driver

|              |                       |
|--------------|-----------------------|
| <i>order</i> | Data order MSB or LSB |
|--------------|-----------------------|

**33.4.5.43** void SAI\_TxSetBitClockPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )

Parameters

|                 |                  |
|-----------------|------------------|
| <i>base</i>     | SAI base pointer |
| <i>polarity</i> |                  |

**33.4.5.44** void SAI\_RxSetBitClockPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )

Parameters

|                 |                  |
|-----------------|------------------|
| <i>base</i>     | SAI base pointer |
| <i>polarity</i> |                  |

**33.4.5.45** void SAI\_TxSetFrameSyncPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )

Parameters

|                 |                  |
|-----------------|------------------|
| <i>base</i>     | SAI base pointer |
| <i>polarity</i> |                  |

**33.4.5.46** void SAI\_RxSetFrameSyncPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )

Parameters

|                 |                  |
|-----------------|------------------|
| <i>base</i>     | SAI base pointer |
| <i>polarity</i> |                  |

**33.4.5.47** `static void SAI_TxEnableInterrupts ( l2s_Type * base, uint32_t mask )`  
`[inline], [static]`

## SAI Driver

### 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"><li>• kSAI_WordStartInterruptEnable</li><li>• kSAI_SyncErrorInterruptEnable</li><li>• kSAI_FIFOWarningInterruptEnable</li><li>• kSAI_FIFOREquestInterruptEnable</li><li>• kSAI_FIFOErrorInterruptEnable</li></ul> |

**33.4.5.48 static void SAI\_RxEnableInterrupts ( 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"><li>• kSAI_WordStartInterruptEnable</li><li>• kSAI_SyncErrorInterruptEnable</li><li>• kSAI_FIFOWarningInterruptEnable</li><li>• kSAI_FIFOREquestInterruptEnable</li><li>• kSAI_FIFOErrorInterruptEnable</li></ul> |

**33.4.5.49 static void SAI\_TxDisableInterrupts ( 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"><li>• kSAI_WordStartInterruptEnable</li><li>• kSAI_SyncErrorInterruptEnable</li><li>• kSAI_FIFOWarningInterruptEnable</li><li>• kSAI_FIFOREquestInterruptEnable</li><li>• kSAI_FIFOErrorInterruptEnable</li></ul> |

### 33.4.5.50 static void SAI\_RxDisableInterrupts ( 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"> <li>• kSAI_WordStartInterruptEnable</li> <li>• kSAI_SyncErrorInterruptEnable</li> <li>• kSAI_FIFOWarningInterruptEnable</li> <li>• kSAI_FIFOREquestInterruptEnable</li> <li>• kSAI_FIFOErrorInterruptEnable</li> </ul> |

### 33.4.5.51 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"> <li>• kSAI_FIFOWarningDMAEnable</li> <li>• kSAI_FIFOREquestDMAEnable</li> </ul> |
| <i>enable</i> | True means enable DMA, false means disable DMA.                                                                                                                                                     |

### 33.4.5.52 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"> <li>• kSAI_FIFOWarningDMAEnable</li> <li>• kSAI_FIFOREquestDMAEnable</li> </ul> |
| <i>enable</i> | True means enable DMA, false means disable DMA.                                                                                                                                                       |

### 33.4.5.53 static uint32\_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.

### 33.4.5.54 static uint32\_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.

### 33.4.5.55 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. |

**33.4.5.56** `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. |

**33.4.5.57** `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.                                     |

## SAI Driver

**33.4.5.58 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.                                     |

**33.4.5.59 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. |

**33.4.5.60 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.                                        |

**33.4.5.61 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.                                        |

**33.4.5.62 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.

**33.4.5.63 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 |

**33.4.5.64** 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. |

### 33.4.5.65 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. |

### 33.4.5.66 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. |

### 33.4.5.67 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.

## SAI Driver

### 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 the status\_t.

**33.4.5.68** `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.

**33.4.5.69** `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.        |

### 33.4.5.70 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 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_RxBusy</i>      | Previous receive still not finished.   |
| <i>kStatus_InvalidArgument</i> | The input parameter is invalid.        |

### 33.4.5.71 status\_t SAI\_TransferGetSendCount ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, size\_t \* *count* )

## SAI Driver

### 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 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. |

### 33.4.5.72 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. |

### 33.4.5.73 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. |



**33.4.5.74 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. |

**33.4.5.75 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. |

**33.4.5.76 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. |

**33.4.5.77 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. |

**33.4.5.78 void SAI\_TransferRxHandleIRQ ( I2S\_Type \* *base*, sai\_handle\_t \* *handle* )**

## SAI Driver

### Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | SAI base pointer.                      |
| <i>handle</i> | Pointer to the sai_handle_t structure. |

## SAI EDMA Driver

### 33.5.1 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, 3, 1))  
Version 2.3.1.

#### 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)

## SAI EDMA Driver

- Performs a non-blocking SAI receive 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.*

### 33.5.2 Data Structure Documentation

#### 33.5.2.1 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](#) [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.*

**33.5.2.1.0.28 Field Documentation****33.5.2.1.0.28.1** `uint8_t sai_edma_handle_t::nbytes`**33.5.2.1.0.28.2** `uint8_t sai_edma_handle_t::tcd[(SAI_XFER_QUEUE_SIZE+1U)*sizeof(edma_tcd_t)]`**33.5.2.1.0.28.3** `sai_transfer_t sai_edma_handle_t::saiQueue[SAI_XFER_QUEUE_SIZE]`**33.5.2.1.0.28.4** `volatile uint8_t sai_edma_handle_t::queueUser`**33.5.3 Function Documentation****33.5.3.1** `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.

## SAI EDMA Driver

### 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. |

**33.5.3.2 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. |

**33.5.3.3 void SAI\_TransferTxSetFormatEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

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. |

**33.5.3.4 void SAI\_TransferRxSetFormatEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

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. |

**33.5.3.5 void SAI\_TransferTxSetConfigEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_transceiver\_t \* *saiConfig* )**

## SAI EDMA Driver

### Parameters

|                  |                          |
|------------------|--------------------------|
| <i>base</i>      | SAI base pointer.        |
| <i>handle</i>    | SAI eDMA handle pointer. |
| <i>saiConfig</i> | sai configurations.      |

**33.5.3.6 void SAI\_TransferRxSetConfigEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_transceiver\_t \* *saiConfig* )**

### Parameters

|                  |                          |
|------------------|--------------------------|
| <i>base</i>      | SAI base pointer.        |
| <i>handle</i>    | SAI eDMA handle pointer. |
| <i>saiConfig</i> | sai configurations.      |

**33.5.3.7 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.

### 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.           |

**33.5.3.8 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.

## 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.            |

### 33.5.3.9 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. |

### 33.5.3.10 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. |
|-------------|-------------------|

## SAI EDMA Driver

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | SAI eDMA handle pointer. |
|---------------|--------------------------|

### 33.5.3.11 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. |

### 33.5.3.12 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. |

### 33.5.3.13 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_NoTransferInProgress</i> | There is no non-blocking transaction in progress. |

**33.5.3.14** `status_t SAI_TransferGetReceiveCountEDMA ( I2S_Type * base,  
sai_edma_handle_t * handle, size_t * count )`

## SAI EDMA Driver

### 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_NoTransferInProgress</i> | There is no non-blocking transaction in progress. |

## Chapter 34

# SDHC: Secure Digital Host Controller Driver

### Overview

The MCUXpresso SDK provides a peripheral driver for the Secure Digital Host Controller (SDHC) module of MCUXpresso SDK devices.

### Typical use case

#### 34.2.1 SDHC Operation

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

### Data Structures

- struct `sdhc_adma2_descriptor_t`  
*Defines the ADMA2 descriptor structure. [More...](#)*
- struct `sdhc_capability_t`  
*SDHC capability information. [More...](#)*
- struct `sdhc_transfer_config_t`  
*Card transfer configuration. [More...](#)*
- struct `sdhc_boot_config_t`  
*Data structure to configure the MMC boot feature. [More...](#)*
- struct `sdhc_config_t`  
*Data structure to initialize the SDHC. [More...](#)*
- struct `sdhc_data_t`  
*Card data descriptor. [More...](#)*
- struct `sdhc_command_t`  
*Card command descriptor. [More...](#)*
- struct `sdhc_transfer_t`  
*Transfer state. [More...](#)*
- struct `sdhc_transfer_callback_t`  
*SDHC callback functions. [More...](#)*
- struct `sdhc_handle_t`  
*SDHC handle. [More...](#)*
- struct `sdhc_host_t`  
*SDHC host descriptor. [More...](#)*

### Macros

- #define `SDHC_MAX_BLOCK_COUNT` (SDHC\_BLKATTR\_BLKCNT\_MASK >> SDHC\_BLKATTR\_BLKCNT\_SHIFT)  
*Maximum block count can be set one time.*
- #define `SDHC_ADMA1_ADDRESS_ALIGN` (4096U)  
*The alignment size for ADDRESS filed in ADMA1's descriptor.*

## Typical use case

- #define `SDHC_ADMA1_LENGTH_ALIGN` (4096U)  
*The alignment size for LENGTH field in ADMA1's descriptor.*
- #define `SDHC_ADMA2_ADDRESS_ALIGN` (4U)  
*The alignment size for ADDRESS field in ADMA2's descriptor.*
- #define `SDHC_ADMA2_LENGTH_ALIGN` (4U)  
*The alignment size for LENGTH field in ADMA2's descriptor.*
- #define `SDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT` (12U)  
*The bit shift for ADDRESS field in ADMA1's descriptor.*
- #define `SDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK` (0xFFFFFU)  
*The bit mask for ADDRESS field in ADMA1's descriptor.*
- #define `SDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT` (12U)  
*The bit shift for LENGTH field in ADMA1's descriptor.*
- #define `SDHC_ADMA1_DESCRIPTOR_LENGTH_MASK` (0xFFFFU)  
*The mask for LENGTH field in ADMA1's descriptor.*
- #define `SDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY` (SDHC\_ADMA1\_DESCRIPTOR\_LENGTH\_MASK + 1U)  
*The maximum value of LENGTH field in ADMA1's descriptor.*
- #define `SDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT` (16U)  
*The bit shift for LENGTH field in ADMA2's descriptor.*
- #define `SDHC_ADMA2_DESCRIPTOR_LENGTH_MASK` (0xFFFFUL)  
*The bit mask for LENGTH field in ADMA2's descriptor.*
- #define `SDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY` (SDHC\_ADMA2\_DESCRIPTOR\_LENGTH\_MASK)  
*The maximum value of LENGTH field in ADMA2's descriptor.*

## Typedefs

- typedef uint32\_t `sdhc_adma1_descriptor_t`  
*Defines the adma1 descriptor structure.*
- typedef `status_t(*sdhc_transfer_function_t)`(SDHC\_Type \*base, `sdhc_transfer_t` \*content)  
*SDHC transfer function.*

## Enumerations

- enum {  
    `kStatus_SDHC_BusyTransferring` = MAKE\_STATUS(kStatusGroup\_SDHC, 0U),  
    `kStatus_SDHC_PrepareAdmaDescriptorFailed` = MAKE\_STATUS(kStatusGroup\_SDHC, 1U),  
    `kStatus_SDHC_SendCommandFailed` = MAKE\_STATUS(kStatusGroup\_SDHC, 2U),  
    `kStatus_SDHC_TransferDataFailed` = MAKE\_STATUS(kStatusGroup\_SDHC, 3U),  
    `kStatus_SDHC_DMADDataBufferAddrNotAlign`,  
    `kStatus_SDHC_TransferCommandComplete` = MAKE\_STATUS(kStatusGroup\_SDHC, 5U),  
    `kStatus_SDHC_TransferDataComplete` = MAKE\_STATUS(kStatusGroup\_SDHC, 6U) }  
    *\_sdhc\_status SDHC status*
- enum {

```

kSDHC_SupportAdmaFlag = SDHC_HTCAPBLT_ADMAS_MASK,
kSDHC_SupportHighSpeedFlag = SDHC_HTCAPBLT_HSS_MASK,
kSDHC_SupportDmaFlag = SDHC_HTCAPBLT_DMAS_MASK,
kSDHC_SupportSuspendResumeFlag = SDHC_HTCAPBLT_SRS_MASK,
kSDHC_SupportV330Flag = SDHC_HTCAPBLT_VS33_MASK,
kSDHC_Support4BitFlag = (SDHC_HTCAPBLT_MBL_SHIFT << 0U),
kSDHC_Support8BitFlag = (SDHC_HTCAPBLT_MBL_SHIFT << 1U) }
 _sdhc_capability_flag Host controller capabilities flag mask
• enum {
 kSDHC_WakeupEventOnCardInt = SDHC_PROCTL_WECINT_MASK,
 kSDHC_WakeupEventOnCardInsert = SDHC_PROCTL_WECINS_MASK,
 kSDHC_WakeupEventOnCardRemove = SDHC_PROCTL_WECRM_MASK,
 kSDHC_WakeupEventsAll }
 _sdhc_wakeup_event Wakeup event mask
• enum {
 kSDHC_ResetAll = SDHC_SYSCTL_RSTA_MASK,
 kSDHC_ResetCommand = SDHC_SYSCTL_RSTC_MASK,
 kSDHC_ResetData = SDHC_SYSCTL_RSTD_MASK,
 kSDHC_ResetsAll = (kSDHC_ResetAll | kSDHC_ResetCommand | kSDHC_ResetData) }
 _sdhc_reset Reset type mask
• enum {
 kSDHC_EnableDmaFlag = SDHC_XFERTYP_DMAEN_MASK,
 kSDHC_CommandTypeSuspendFlag = (SDHC_XFERTYP_CMDTYP(1U)),
 kSDHC_CommandTypeResumeFlag = (SDHC_XFERTYP_CMDTYP(2U)),
 kSDHC_CommandTypeAbortFlag = (SDHC_XFERTYP_CMDTYP(3U)),
 kSDHC_EnableBlockCountFlag = SDHC_XFERTYP_BCEN_MASK,
 kSDHC_EnableAutoCommand12Flag = SDHC_XFERTYP_AC12EN_MASK,
 kSDHC_DataReadFlag = SDHC_XFERTYP_DTDSEL_MASK,
 kSDHC_MultipleBlockFlag = SDHC_XFERTYP_MSBSEL_MASK,
 kSDHC_ResponseLength136Flag = SDHC_XFERTYP_RSPTYP(1U),
 kSDHC_ResponseLength48Flag = SDHC_XFERTYP_RSPTYP(2U),
 kSDHC_ResponseLength48BusyFlag = SDHC_XFERTYP_RSPTYP(3U),
 kSDHC_EnableCrcCheckFlag = SDHC_XFERTYP_CCCEN_MASK,
 kSDHC_EnableIndexCheckFlag = SDHC_XFERTYP_CICEN_MASK,
 kSDHC_DataPresentFlag = SDHC_XFERTYP_DPSEL_MASK }
 _sdhc_transfer_flag Transfer flag mask
• enum {

```

## Typical use case

```
kSDHC_CommandInhibitFlag = SDHC_PRSTAT_CIHB_MASK,
kSDHC_DataInhibitFlag = SDHC_PRSTAT_CDIHB_MASK,
kSDHC_DataLineActiveFlag = SDHC_PRSTAT_DLA_MASK,
kSDHC_SdClockStableFlag = SDHC_PRSTAT_SDSTB_MASK,
kSDHC_WriteTransferActiveFlag = SDHC_PRSTAT_WTA_MASK,
kSDHC_ReadTransferActiveFlag = SDHC_PRSTAT_RTA_MASK,
kSDHC_BufferWriteEnableFlag = SDHC_PRSTAT_BWEN_MASK,
kSDHC_BufferReadEnableFlag = SDHC_PRSTAT_BREN_MASK,
kSDHC_CardInsertedFlag = SDHC_PRSTAT_CINS_MASK,
kSDHC_CommandLineLevelFlag = SDHC_PRSTAT_CLSL_MASK,
kSDHC_Data0LineLevelFlag = (1U << 24U),
kSDHC_Data1LineLevelFlag = (1U << 25U),
kSDHC_Data2LineLevelFlag = (1U << 26U),
kSDHC_Data3LineLevelFlag = (1U << 27U),
kSDHC_Data4LineLevelFlag = (1U << 28U),
kSDHC_Data5LineLevelFlag = (1U << 29U),
kSDHC_Data6LineLevelFlag = (1U << 30U),
kSDHC_Data7LineLevelFlag = (int)(1U << 31U) }
_sdhc_present_status_flag Present status flag mask
• enum {
kSDHC_CommandCompleteFlag = SDHC_IRQSTAT_CC_MASK,
kSDHC_DataCompleteFlag = SDHC_IRQSTAT_TC_MASK,
kSDHC_BlockGapEventFlag = SDHC_IRQSTAT_BGE_MASK,
kSDHC_DmaCompleteFlag = SDHC_IRQSTAT_DINT_MASK,
kSDHC_BufferWriteReadyFlag = SDHC_IRQSTAT_BWR_MASK,
kSDHC_BufferReadReadyFlag = SDHC_IRQSTAT_BRR_MASK,
kSDHC_CardInsertionFlag = SDHC_IRQSTAT_CINS_MASK,
kSDHC_CardRemovalFlag = SDHC_IRQSTAT_CRM_MASK,
kSDHC_CardInterruptFlag = SDHC_IRQSTAT_CINT_MASK,
kSDHC_CommandTimeoutFlag = SDHC_IRQSTAT_CTOE_MASK,
kSDHC_CommandCrcErrorFlag = SDHC_IRQSTAT_CCE_MASK,
kSDHC_CommandEndBitErrorFlag = SDHC_IRQSTAT_CEBE_MASK,
kSDHC_CommandIndexErrorFlag = SDHC_IRQSTAT_CIE_MASK,
kSDHC_DataTimeoutFlag = SDHC_IRQSTAT_DTOE_MASK,
kSDHC_DataCrcErrorFlag = SDHC_IRQSTAT_DCE_MASK,
kSDHC_DataEndBitErrorFlag = SDHC_IRQSTAT_DEBE_MASK,
kSDHC_AutoCommand12ErrorFlag = SDHC_IRQSTAT_AC12E_MASK,
kSDHC_DmaErrorFlag = SDHC_IRQSTAT_DMAE_MASK,
kSDHC_CommandErrorFlag,
kSDHC_DataErrorFlag,
kSDHC_ErrorFlag = (kSDHC_CommandErrorFlag | kSDHC_DataErrorFlag | kSDHC_DmaError-
Flag),
kSDHC_DataFlag,
kSDHC_DataDMAFlag = (kSDHC_DataCompleteFlag | kSDHC_DataErrorFlag | kSDHC_Dma-
```



```

ErrorFlag),
kSDHC_CommandFlag = (kSDHC_CommandErrorFlag | kSDHC_CommandCompleteFlag),
kSDHC_CardDetectFlag = (kSDHC_CardInsertionFlag | kSDHC_CardRemovalFlag),
kSDHC_AllInterruptFlags }
 _sdhc_interrupt_status_flag Interrupt status flag mask
• enum {
 kSDHC_AutoCommand12NotExecutedFlag = SDHC_AC12ERR_AC12NE_MASK,
 kSDHC_AutoCommand12TimeoutFlag = SDHC_AC12ERR_AC12TOE_MASK,
 kSDHC_AutoCommand12EndBitErrorFlag = SDHC_AC12ERR_AC12EBE_MASK,
 kSDHC_AutoCommand12CrcErrorFlag = SDHC_AC12ERR_AC12CE_MASK,
 kSDHC_AutoCommand12IndexErrorFlag = SDHC_AC12ERR_AC12IE_MASK,
 kSDHC_AutoCommand12NotIssuedFlag = SDHC_AC12ERR_CNIBAC12E_MASK }
 _sdhc_auto_command12_error_status_flag Auto CMD12 error status flag mask
• enum {
 kSDHC_AdmaLenghMismatchFlag = SDHC_ADMAES_ADMALME_MASK,
 kSDHC_AdmaDescriptorErrorFlag = SDHC_ADMAES_ADMADCE_MASK }
 _sdhc_adma_error_status_flag ADMA error status flag mask
• enum sdhc_adma_error_state_t {
 kSDHC_AdmaErrorStateStopDma = 0x00U,
 kSDHC_AdmaErrorStateFetchDescriptor = 0x01U,
 kSDHC_AdmaErrorStateChangeAddress = 0x02U,
 kSDHC_AdmaErrorStateTransferData = 0x03U }
 ADMA error state.
• enum {
 kSDHC_ForceEventAutoCommand12NotExecuted = SDHC_FEVT_AC12NE_MASK,
 kSDHC_ForceEventAutoCommand12Timeout = SDHC_FEVT_AC12TOE_MASK,
 kSDHC_ForceEventAutoCommand12CrcError = SDHC_FEVT_AC12CE_MASK,
 kSDHC_ForceEventEndBitError = SDHC_FEVT_AC12EBE_MASK,
 kSDHC_ForceEventAutoCommand12IndexError = SDHC_FEVT_AC12IE_MASK,
 kSDHC_ForceEventAutoCommand12NotIssued = SDHC_FEVT_CNIBAC12E_MASK,
 kSDHC_ForceEventCommandTimeout = SDHC_FEVT_CTOE_MASK,
 kSDHC_ForceEventCommandCrcError = SDHC_FEVT_CCE_MASK,
 kSDHC_ForceEventCommandEndBitError = SDHC_FEVT_CEBE_MASK,
 kSDHC_ForceEventCommandIndexError = SDHC_FEVT_CIE_MASK,
 kSDHC_ForceEventDataTimeout = SDHC_FEVT_DTOE_MASK,
 kSDHC_ForceEventDataCrcError = SDHC_FEVT_DCE_MASK,
 kSDHC_ForceEventDataEndBitError = SDHC_FEVT_DEBE_MASK,
 kSDHC_ForceEventAutoCommand12Error = SDHC_FEVT_AC12E_MASK,
 kSDHC_ForceEventCardInt = (int)SDHC_FEVT_CINT_MASK,
 kSDHC_ForceEventDmaError = SDHC_FEVT_DMAE_MASK,
 kSDHC_ForceEventsAll }
 _sdhc_force_event Force event bit position
• enum sdhc_data_bus_width_t {
 kSDHC_DataBusWidth1Bit = 0U,
 kSDHC_DataBusWidth4Bit = 1U,
 kSDHC_DataBusWidth8Bit = 2U }

```

## Typical use case

- Data transfer width.*
  - enum `sdhc_endian_mode_t` {  
    `kSDHC_EndianModeBig` = 0U,  
    `kSDHC_EndianModeHalfWordBig` = 1U,  
    `kSDHC_EndianModeLittle` = 2U }
- Endian mode.*
  - enum `sdhc_dma_mode_t` {  
    `kSDHC_DmaModeNo` = 0U,  
    `kSDHC_DmaModeAdma1` = 1U,  
    `kSDHC_DmaModeAdma2` = 2U }
- DMA mode.*
  - enum {  
    `kSDHC_StopAtBlockGapFlag` = 0x01,  
    `kSDHC_ReadWaitControlFlag` = 0x02,  
    `kSDHC_InterruptAtBlockGapFlag` = 0x04,  
    `kSDHC_ExactBlockNumberReadFlag` = 0x08 }
  - \_sdhc\_sdio\_control\_flag SDIO control flag mask*
  - enum `sdhc_boot_mode_t` {  
    `kSDHC_BootModeNormal` = 0U,  
    `kSDHC_BootModeAlternative` = 1U }
  - MMC card boot mode.*
    - enum `sdhc_card_command_type_t` {  
    `kCARD_CommandTypeNormal` = 0U,  
    `kCARD_CommandTypeSuspend` = 1U,  
    `kCARD_CommandTypeResume` = 2U,  
    `kCARD_CommandTypeAbort` = 3U }
  - The command type.*
    - enum `sdhc_card_response_type_t` {  
    `kCARD_ResponseTypeNone` = 0U,  
    `kCARD_ResponseTypeR1` = 1U,  
    `kCARD_ResponseTypeR1b` = 2U,  
    `kCARD_ResponseTypeR2` = 3U,  
    `kCARD_ResponseTypeR3` = 4U,  
    `kCARD_ResponseTypeR4` = 5U,  
    `kCARD_ResponseTypeR5` = 6U,  
    `kCARD_ResponseTypeR5b` = 7U,  
    `kCARD_ResponseTypeR6` = 8U,  
    `kCARD_ResponseTypeR7` = 9U }
  - The command response type.*
    - enum {

- ```

kSDHC_Adma1DescriptorValidFlag = (1U << 0U),
kSDHC_Adma1DescriptorEndFlag = (1U << 1U),
kSDHC_Adma1DescriptorInterruptFlag = (1U << 2U),
kSDHC_Adma1DescriptorActivity1Flag = (1U << 4U),
kSDHC_Adma1DescriptorActivity2Flag = (1U << 5U),
kSDHC_Adma1DescriptorTypeNop = (kSDHC_Adma1DescriptorValidFlag),
kSDHC_Adma1DescriptorTypeTransfer,
kSDHC_Adma1DescriptorTypeLink,
kSDHC_Adma1DescriptorTypeSetLength }
    _sdhc_adma1_descriptor_flag The mask for the control/status field in ADMA1 descriptor

```
- enum {

```

kSDHC_Adma2DescriptorValidFlag = (1U << 0U),
kSDHC_Adma2DescriptorEndFlag = (1U << 1U),
kSDHC_Adma2DescriptorInterruptFlag = (1U << 2U),
kSDHC_Adma2DescriptorActivity1Flag = (1U << 4U),
kSDHC_Adma2DescriptorActivity2Flag = (1U << 5U),
kSDHC_Adma2DescriptorTypeNop = (kSDHC_Adma2DescriptorValidFlag),
kSDHC_Adma2DescriptorTypeReserved,
kSDHC_Adma2DescriptorTypeTransfer,
kSDHC_Adma2DescriptorTypeLink }
    _sdhc_adma2_descriptor_flag ADMA1 descriptor control and status mask

```

Driver version

- #define **FSL_SDHC_DRIVER_VERSION** (MAKE_VERSION(2U, 1U, 11U))
Driver version 2.1.11.

Initialization and deinitialization

- void **SDHC_Init** (SDHC_Type *base, const **sdhc_config_t** *config)
SDHC module initialization function.
- void **SDHC_Deinit** (SDHC_Type *base)
Deinitializes the SDHC.
- bool **SDHC_Reset** (SDHC_Type *base, uint32_t mask, uint32_t timeout)
Resets the SDHC.

DMA Control

- status_t** **SDHC_SetAdmaTableConfig** (SDHC_Type *base, **sdhc_dma_mode_t** dmaMode, uint32_t *table, uint32_t tableWords, const uint32_t *data, uint32_t dataBytes)
Sets the ADMA descriptor table configuration.

Interrupts

- static void **SDHC_EnableInterruptStatus** (SDHC_Type *base, uint32_t mask)
Enables the interrupt status.
- static void **SDHC_DisableInterruptStatus** (SDHC_Type *base, uint32_t mask)
Disables the interrupt status.

Typical use case

- static void [SDHC_EnableInterruptSignal](#) (SDHC_Type *base, uint32_t mask)
Enables the interrupt signal corresponding to the interrupt status flag.
- static void [SDHC_DisableInterruptSignal](#) (SDHC_Type *base, uint32_t mask)
Disables the interrupt signal corresponding to the interrupt status flag.

Status

- static uint32_t [SDHC_GetEnabledInterruptStatusFlags](#) (SDHC_Type *base)
Gets the enabled interrupt status.
- static uint32_t [SDHC_GetInterruptStatusFlags](#) (SDHC_Type *base)
Gets the current interrupt status.
- static void [SDHC_ClearInterruptStatusFlags](#) (SDHC_Type *base, uint32_t mask)
Clears a specified interrupt status.
- static uint32_t [SDHC_GetAutoCommand12ErrorStatusFlags](#) (SDHC_Type *base)
Gets the status of auto command 12 error.
- static uint32_t [SDHC_GetAdmaErrorStatusFlags](#) (SDHC_Type *base)
Gets the status of the ADMA error.
- static uint32_t [SDHC_GetPresentStatusFlags](#) (SDHC_Type *base)
Gets a present status.

Bus Operations

- void [SDHC_GetCapability](#) (SDHC_Type *base, [sdhc_capability_t](#) *capability)
Gets the capability information.
- static void [SDHC_EnableSdClock](#) (SDHC_Type *base, bool enable)
Enables or disables the SD bus clock.
- uint32_t [SDHC_SetSdClock](#) (SDHC_Type *base, uint32_t srcClock_Hz, uint32_t busClock_Hz)
Sets the SD bus clock frequency.
- bool [SDHC_SetCardActive](#) (SDHC_Type *base, uint32_t timeout)
Sends 80 clocks to the card to set it to the active state.
- static void [SDHC_SetDataBusWidth](#) (SDHC_Type *base, [sdhc_data_bus_width_t](#) width)
Sets the data transfer width.
- static void [SDHC_CardDetectByData3](#) (SDHC_Type *base, bool enable)
detect card insert status.
- void [SDHC_SetTransferConfig](#) (SDHC_Type *base, const [sdhc_transfer_config_t](#) *config)
Sets the card transfer-related configuration.
- static uint32_t [SDHC_GetCommandResponse](#) (SDHC_Type *base, uint32_t index)
Gets the command response.
- static void [SDHC_WriteData](#) (SDHC_Type *base, uint32_t data)
Fills the data port.
- static uint32_t [SDHC_ReadData](#) (SDHC_Type *base)
Retrieves the data from the data port.
- static void [SDHC_EnableWakeupEvent](#) (SDHC_Type *base, uint32_t mask, bool enable)
Enables or disables a wakeup event in low-power mode.
- static void [SDHC_EnableCardDetectTest](#) (SDHC_Type *base, bool enable)
Enables or disables the card detection level for testing.
- static void [SDHC_SetCardDetectTestLevel](#) (SDHC_Type *base, bool high)
Sets the card detection test level.
- void [SDHC_EnableSdioControl](#) (SDHC_Type *base, uint32_t mask, bool enable)
Enables or disables the SDIO card control.
- static void [SDHC_SetContinueRequest](#) (SDHC_Type *base)

- Restarts a transaction which has stopped at the block GAP for the SDIO card.
- void [SDHC_SetMmcBootConfig](#) (SDHC_Type *base, const [sdhc_boot_config_t](#) *config)
Configures the MMC boot feature.
- static void [SDHC_SetForceEvent](#) (SDHC_Type *base, uint32_t mask)
Forces generating events according to the given mask.

Transactional

- [status_t SDHC_TransferBlocking](#) (SDHC_Type *base, uint32_t *admaTable, uint32_t admaTableWords, [sdhc_transfer_t](#) *transfer)
Transfers the command/data using a blocking method.
- void [SDHC_TransferCreateHandle](#) (SDHC_Type *base, [sdhc_handle_t](#) *handle, const [sdhc_transfer_callback_t](#) *callback, void *userData)
Creates the SDHC handle.
- [status_t SDHC_TransferNonBlocking](#) (SDHC_Type *base, [sdhc_handle_t](#) *handle, uint32_t *admaTable, uint32_t admaTableWords, [sdhc_transfer_t](#) *transfer)
Transfers the command/data using an interrupt and an asynchronous method.
- void [SDHC_TransferHandleIRQ](#) (SDHC_Type *base, [sdhc_handle_t](#) *handle)
IRQ handler for the SDHC.

Data Structure Documentation

34.3.1 struct sdhc_adma2_descriptor_t

Data Fields

- uint32_t [attribute](#)
The control and status field.
- const uint32_t * [address](#)
The address field.

34.3.2 struct sdhc_capability_t

Defines a structure to save the capability information of SDHC.

Data Fields

- uint32_t [specVersion](#)
Specification version.
- uint32_t [vendorVersion](#)
Vendor version.
- uint32_t [maxBlockLength](#)
Maximum block length united as byte.
- uint32_t [maxBlockCount](#)
Maximum block count can be set one time.
- uint32_t [flags](#)
Capability flags to indicate the support information([_sdhc_capability_flag](#))

34.3.3 struct sdhc_transfer_config_t

Define structure to configure the transfer-related command index/argument/flags and data block size/data block numbers. This structure needs to be filled each time a command is sent to the card.

Data Fields

- size_t [dataBlockSize](#)
Data block size.
- uint32_t [dataBlockCount](#)
Data block count.
- uint32_t [commandArgument](#)
Command argument.
- uint32_t [commandIndex](#)
Command index.
- uint32_t [flags](#)
Transfer flags(_sdhc_transfer_flag)

34.3.4 struct sdhc_boot_config_t

Data Fields

- uint32_t [ackTimeoutCount](#)
Timeout value for the boot ACK.
- [sdhc_boot_mode_t](#) [bootMode](#)
Boot mode selection.
- uint32_t [blockCount](#)
Stop at block gap value of automatic mode.
- bool [enableBootAck](#)
Enable or disable boot ACK.
- bool [enableBoot](#)
Enable or disable fast boot.
- bool [enableAutoStopAtBlockGap](#)
Enable or disable auto stop at block gap function in boot period.

34.3.4.0.0.29 Field Documentation

34.3.4.0.0.29.1 uint32_t sdhc_boot_config_t::ackTimeoutCount

The available range is 0 ~ 15.

34.3.4.0.0.29.2 sdhc_boot_mode_t sdhc_boot_config_t::bootMode

34.3.4.0.0.29.3 uint32_t sdhc_boot_config_t::blockCount

Available range is 0 ~ 65535.

34.3.5 struct sdhc_config_t

Data Fields

- bool [cardDetectDat3](#)
Enable DAT3 as card detection pin.
- [sdhc_endian_mode_t](#) [endianMode](#)
Endian mode.
- [sdhc_dma_mode_t](#) [dmaMode](#)
DMA mode.
- uint32_t [readWatermarkLevel](#)
Watermark level for DMA read operation.
- uint32_t [writeWatermarkLevel](#)
Watermark level for DMA write operation.

34.3.5.0.0.30 Field Documentation

34.3.5.0.0.30.1 uint32_t sdhc_config_t::readWatermarkLevel

Available range is 1 ~ 128.

34.3.5.0.0.30.2 uint32_t sdhc_config_t::writeWatermarkLevel

Available range is 1 ~ 128.

34.3.6 struct sdhc_data_t

Defines a structure to contain data-related attribute. 'enableIgnoreError' is used for the case that upper card driver want to ignore the error event to read/write all the data not to stop read/write immediately when error event happen for example bus testing procedure for MMC card.

Data Fields

- bool [enableAutoCommand12](#)
Enable auto CMD12.
- bool [enableIgnoreError](#)
Enable to ignore error event to read/write all the data.
- size_t [blockSize](#)
Block size.
- uint32_t [blockCount](#)
Block count.
- uint32_t * [rxData](#)
Buffer to save data read.
- const uint32_t * [txData](#)
Data buffer to write.

34.3.7 struct sdhc_command_t

Define card command-related attribute.

Data Fields

- uint32_t [index](#)
Command index.
- uint32_t [argument](#)
Command argument.
- [sdhc_card_command_type_t](#) type
Command type.
- [sdhc_card_response_type_t](#) responseType
Command response type.
- uint32_t [response](#) [4U]
Response for this command.
- uint32_t [responseErrorFlags](#)
response error flag, the flag which need to check the command reponse

34.3.8 struct sdhc_transfer_t

Data Fields

- [sdhc_data_t](#) * data
Data to transfer.
- [sdhc_command_t](#) * command
Command to send.

34.3.9 struct sdhc_transfer_callback_t

Data Fields

- void(* [CardInserted](#))(SDHC_Type *base, void *userData)
Card inserted occurs when DAT3/CD pin is for card detect.
- void(* [CardRemoved](#))(SDHC_Type *base, void *userData)
Card removed occurs.
- void(* [SdioInterrupt](#))(SDHC_Type *base, void *userData)
SDIO card interrupt occurs.
- void(* [SdioBlockGap](#))(SDHC_Type *base, void *userData)
SDIO card stopped at block gap occurs.
- void(* [TransferComplete](#))(SDHC_Type *base, sdhc_handle_t *handle, [status_t](#) status, void *userData)
Transfer complete callback.

34.3.10 struct _sdhc_handle

SDHC handle typedef.

Defines the structure to save the SDHC state information and callback function. The detailed interrupt status when sending a command or transferring data can be obtained from the interruptFlags field by using the mask defined in sdhc_interrupt_flag_t.

Note

All the fields except interruptFlags and transferredWords must be allocated by the user.

Data Fields

- [sdhc_data_t](#) *volatile data
Data to transfer.
- [sdhc_command_t](#) *volatile command
Command to send.
- volatile uint32_t transferredWords
Words transferred by DATAPORT way.
- [sdhc_transfer_callback_t](#) callback
Callback function.
- void * userData
Parameter for transfer complete callback.

34.3.11 struct sdhc_host_t

Data Fields

- SDHC_Type * base
SDHC peripheral base address.
- uint32_t sourceClock_Hz
SDHC source clock frequency united in Hz.
- [sdhc_config_t](#) config
SDHC configuration.
- [sdhc_capability_t](#) capability
SDHC capability information.
- [sdhc_transfer_function_t](#) transfer
SDHC transfer function.

Macro Definition Documentation

34.4.1 #define FSL_SDHC_DRIVER_VERSION (MAKE_VERSION(2U, 1U, 11U))

Enumeration Type Documentation

Typedef Documentation

34.5.1 typedef uint32_t sdhc_adma1_descriptor_t

34.5.2 typedef status_t(* sdhc_transfer_function_t)(SDHC_Type *base, sdhc_transfer_t *content)

Enumeration Type Documentation

34.6.1 anonymous enum

Enumerator

kStatus_SDHC_BusyTransferring Transfer is on-going.
kStatus_SDHC_PrepareAdmaDescriptorFailed Set DMA descriptor failed.
kStatus_SDHC_SendCommandFailed Send command failed.
kStatus_SDHC_TransferDataFailed Transfer data failed.
kStatus_SDHC_DMADDataBufferAddrNotAlign data buffer addr not align in DMA mode
kStatus_SDHC_TransferCommandComplete command transfer complete
kStatus_SDHC_TransferDataComplete data transfer complete

34.6.2 anonymous enum

Enumerator

kSDHC_SupportAdmaFlag Support ADMA.
kSDHC_SupportHighSpeedFlag Support high-speed.
kSDHC_SupportDmaFlag Support DMA.
kSDHC_SupportSuspendResumeFlag Support suspend/resume.
kSDHC_SupportV330Flag Support voltage 3.3V.
kSDHC_Support4BitFlag Support 4 bit mode.
kSDHC_Support8BitFlag Support 8 bit mode.

34.6.3 anonymous enum

Enumerator

kSDHC_WakeupEventOnCardInt Wakeup on card interrupt.
kSDHC_WakeupEventOnCardInsert Wakeup on card insertion.
kSDHC_WakeupEventOnCardRemove Wakeup on card removal.
kSDHC_WakeupEventsAll All wakeup events.

34.6.4 anonymous enum

Enumerator

kSDHC_ResetAll Reset all except card detection.

kSDHC_ResetCommand Reset command line.

kSDHC_ResetData Reset data line.

kSDHC_ResetsAll All reset types.

34.6.5 anonymous enum

Enumerator

kSDHC_EnableDmaFlag Enable DMA.

kSDHC_CommandTypeSuspendFlag Suspend command.

kSDHC_CommandTypeResumeFlag Resume command.

kSDHC_CommandTypeAbortFlag Abort command.

kSDHC_EnableBlockCountFlag Enable block count.

kSDHC_EnableAutoCommand12Flag Enable auto CMD12.

kSDHC_DataReadFlag Enable data read.

kSDHC_MultipleBlockFlag Multiple block data read/write.

kSDHC_ResponseLength136Flag 136 bit response length

kSDHC_ResponseLength48Flag 48 bit response length

kSDHC_ResponseLength48BusyFlag 48 bit response length with busy status

kSDHC_EnableCrcCheckFlag Enable CRC check.

kSDHC_EnableIndexCheckFlag Enable index check.

kSDHC_DataPresentFlag Data present flag.

34.6.6 anonymous enum

Enumerator

kSDHC_CommandInhibitFlag Command inhibit.

kSDHC_DataInhibitFlag Data inhibit.

kSDHC_DataLineActiveFlag Data line active.

kSDHC_SdClockStableFlag SD bus clock stable.

kSDHC_WriteTransferActiveFlag Write transfer active.

kSDHC_ReadTransferActiveFlag Read transfer active.

kSDHC_BufferWriteEnableFlag Buffer write enable.

kSDHC_BufferReadEnableFlag Buffer read enable.

kSDHC_CardInsertedFlag Card inserted.

kSDHC_CommandLineLevelFlag Command line signal level.

kSDHC_Data0LineLevelFlag Data0 line signal level.

kSDHC_Data1LineLevelFlag Data1 line signal level.

Enumeration Type Documentation

kSDHC_Data2LineLevelFlag Data2 line signal level.
kSDHC_Data3LineLevelFlag Data3 line signal level.
kSDHC_Data4LineLevelFlag Data4 line signal level.
kSDHC_Data5LineLevelFlag Data5 line signal level.
kSDHC_Data6LineLevelFlag Data6 line signal level.
kSDHC_Data7LineLevelFlag Data7 line signal level.

34.6.7 anonymous enum

Enumerator

kSDHC_CommandCompleteFlag Command complete.
kSDHC_DataCompleteFlag Data complete.
kSDHC_BlockGapEventFlag Block gap event.
kSDHC_DmaCompleteFlag DMA interrupt.
kSDHC_BufferWriteReadyFlag Buffer write ready.
kSDHC_BufferReadReadyFlag Buffer read ready.
kSDHC_CardInsertionFlag Card inserted.
kSDHC_CardRemovalFlag Card removed.
kSDHC_CardInterruptFlag Card interrupt.
kSDHC_CommandTimeoutFlag Command timeout error.
kSDHC_CommandCrcErrorFlag Command CRC error.
kSDHC_CommandEndBitErrorFlag Command end bit error.
kSDHC_CommandIndexErrorFlag Command index error.
kSDHC_DataTimeoutFlag Data timeout error.
kSDHC_DataCrcErrorFlag Data CRC error.
kSDHC_DataEndBitErrorFlag Data end bit error.
kSDHC_AutoCommand12ErrorFlag Auto CMD12 error.
kSDHC_DmaErrorFlag DMA error.
kSDHC_CommandErrorFlag Command error.
kSDHC_DataErrorFlag Data error.
kSDHC_ErrorFlag All error.
kSDHC_DataFlag Data interrupts.
kSDHC_DataDMAFlag Data interrupts.
kSDHC_CommandFlag Command interrupts.
kSDHC_CardDetectFlag Card detection interrupts.
kSDHC_AllInterruptFlags All flags mask.

34.6.8 anonymous enum

Enumerator

kSDHC_AutoCommand12NotExecutedFlag Not executed error.
kSDHC_AutoCommand12TimeoutFlag Timeout error.

kSDHC_AutoCommand12EndBitErrorFlag End bit error.
kSDHC_AutoCommand12CrcErrorFlag CRC error.
kSDHC_AutoCommand12IndexErrorFlag Index error.
kSDHC_AutoCommand12NotIssuedFlag Not issued error.

34.6.9 anonymous enum

Enumerator

kSDHC_AdmaLenghMismatchFlag Length mismatch error.
kSDHC_AdmaDescriptorErrorFlag Descriptor error.

34.6.10 enum sdhc_adma_error_state_t

This state is the detail state when ADMA error has occurred.

Enumerator

kSDHC_AdmaErrorStateStopDma Stop DMA.
kSDHC_AdmaErrorStateFetchDescriptor Fetch descriptor.
kSDHC_AdmaErrorStateChangeAddress Change address.
kSDHC_AdmaErrorStateTransferData Transfer data.

34.6.11 anonymous enum

Enumerator

kSDHC_ForceEventAutoCommand12NotExecuted Auto CMD12 not executed error.
kSDHC_ForceEventAutoCommand12Timeout Auto CMD12 timeout error.
kSDHC_ForceEventAutoCommand12CrcError Auto CMD12 CRC error.
kSDHC_ForceEventEndBitError Auto CMD12 end bit error.
kSDHC_ForceEventAutoCommand12IndexError Auto CMD12 index error.
kSDHC_ForceEventAutoCommand12NotIssued Auto CMD12 not issued error.
kSDHC_ForceEventCommandTimeout Command timeout error.
kSDHC_ForceEventCommandCrcError Command CRC error.
kSDHC_ForceEventCommandEndBitError Command end bit error.
kSDHC_ForceEventCommandIndexError Command index error.
kSDHC_ForceEventDataTimeout Data timeout error.
kSDHC_ForceEventDataCrcError Data CRC error.
kSDHC_ForceEventDataEndBitError Data end bit error.
kSDHC_ForceEventAutoCommand12Error Auto CMD12 error.
kSDHC_ForceEventCardInt Card interrupt.
kSDHC_ForceEventDmaError Dma error.

Enumeration Type Documentation

kSDHC_ForceEventsAll All force event flags mask.

34.6.12 enum sdhc_data_bus_width_t

Enumerator

kSDHC_DataBusWidth1Bit 1-bit mode

kSDHC_DataBusWidth4Bit 4-bit mode

kSDHC_DataBusWidth8Bit 8-bit mode

34.6.13 enum sdhc_endian_mode_t

Enumerator

kSDHC_EndianModeBig Big endian mode.

kSDHC_EndianModeHalfWordBig Half word big endian mode.

kSDHC_EndianModeLittle Little endian mode.

34.6.14 enum sdhc_dma_mode_t

Enumerator

kSDHC_DmaModeNo No DMA.

kSDHC_DmaModeAdma1 ADMA1 is selected.

kSDHC_DmaModeAdma2 ADMA2 is selected.

34.6.15 anonymous enum

Enumerator

kSDHC_StopAtBlockGapFlag Stop at block gap.

kSDHC_ReadWaitControlFlag Read wait control.

kSDHC_InterruptAtBlockGapFlag Interrupt at block gap.

kSDHC_ExactBlockNumberReadFlag Exact block number read.

34.6.16 enum sdhc_boot_mode_t

Enumerator

kSDHC_BootModeNormal Normal boot.

kSDHC_BootModeAlternative Alternative boot.

34.6.17 enum sdhc_card_command_type_t

Enumerator

kCARD_CommandTypeNormal Normal command.
kCARD_CommandTypeSuspend Suspend command.
kCARD_CommandTypeResume Resume command.
kCARD_CommandTypeAbort Abort command.

34.6.18 enum sdhc_card_response_type_t

Define the command response type from card to host controller.

Enumerator

kCARD_ResponseTypeNone Response type: none.
kCARD_ResponseTypeR1 Response type: R1.
kCARD_ResponseTypeR1b Response type: R1b.
kCARD_ResponseTypeR2 Response type: R2.
kCARD_ResponseTypeR3 Response type: R3.
kCARD_ResponseTypeR4 Response type: R4.
kCARD_ResponseTypeR5 Response type: R5.
kCARD_ResponseTypeR5b Response type: R5b.
kCARD_ResponseTypeR6 Response type: R6.
kCARD_ResponseTypeR7 Response type: R7.

34.6.19 anonymous enum

Enumerator

kSDHC_Adma1DescriptorValidFlag Valid flag.
kSDHC_Adma1DescriptorEndFlag End flag.
kSDHC_Adma1DescriptorInterruptFlag Interrupt flag.
kSDHC_Adma1DescriptorActivity1Flag Activity 1 flag.
kSDHC_Adma1DescriptorActivity2Flag Activity 2 flag.
kSDHC_Adma1DescriptorTypeNop No operation.
kSDHC_Adma1DescriptorTypeTransfer Transfer data.
kSDHC_Adma1DescriptorTypeLink Link descriptor.
kSDHC_Adma1DescriptorTypeSetLength Set data length.

Function Documentation

34.6.20 anonymous enum

Enumerator

kSDHC_Adma2DescriptorValidFlag Valid flag.
kSDHC_Adma2DescriptorEndFlag End flag.
kSDHC_Adma2DescriptorInterruptFlag Interrupt flag.
kSDHC_Adma2DescriptorActivity1Flag Activity 1 mask.
kSDHC_Adma2DescriptorActivity2Flag Activity 2 mask.
kSDHC_Adma2DescriptorTypeNop No operation.
kSDHC_Adma2DescriptorTypeReserved Reserved.
kSDHC_Adma2DescriptorTypeTransfer Transfer type.
kSDHC_Adma2DescriptorTypeLink Link type.

Function Documentation

34.7.1 void SDHC_Init (SDHC_Type * *base*, const sdhc_config_t * *config*)

Configures the SDHC according to the user configuration.

Example:

```
sdhc_config_t config;  
config.cardDetectDat3 = false;  
config.endianMode = kSDHC_EndianModeLittle;  
config.dmaMode = kSDHC_DmaModeAdma2;  
config.readWatermarkLevel = 128U;  
config.writeWatermarkLevel = 128U;  
SDHC_Init(SDHC, &config);
```

Parameters

<i>base</i>	SDHC peripheral base address.
<i>config</i>	SDHC configuration information.

Return values

<i>kStatus_Success</i>	Operate successfully.
------------------------	-----------------------

34.7.2 void SDHC_Deinit (SDHC_Type * *base*)

Parameters

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

34.7.3 bool SDHC_Reset (SDHC_Type * *base*, uint32_t *mask*, uint32_t *timeout*)

Parameters

<i>base</i>	SDHC peripheral base address.
<i>mask</i>	The reset type mask(_sdhc_reset).
<i>timeout</i>	Timeout for reset.

Return values

<i>true</i>	Reset successfully.
<i>false</i>	Reset failed.

34.7.4 status_t SDHC_SetAdmaTableConfig (SDHC_Type * *base*, sdhc_dma_mode_t *dmaMode*, uint32_t * *table*, uint32_t *tableWords*, const uint32_t * *data*, uint32_t *dataBytes*)

Parameters

<i>base</i>	SDHC peripheral base address.
<i>dmaMode</i>	DMA mode.
<i>table</i>	ADMA table address.
<i>tableWords</i>	ADMA table buffer length united as Words.
<i>data</i>	Data buffer address.
<i>dataBytes</i>	Data length united as bytes.

Return values

<i>kStatus_OutOfRange</i>	ADMA descriptor table length isn't enough to describe data.
---------------------------	---

Function Documentation

<i>kStatus_Success</i>	Operate successfully.
------------------------	-----------------------

34.7.5 static void SDHC_EnableInterruptStatus (SDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SDHC peripheral base address.
<i>mask</i>	Interrupt status flags mask(_sdhc_interrupt_status_flag).

34.7.6 static void SDHC_DisableInterruptStatus (SDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SDHC peripheral base address.
<i>mask</i>	The interrupt status flags mask(_sdhc_interrupt_status_flag).

34.7.7 static void SDHC_EnableInterruptSignal (SDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SDHC peripheral base address.
<i>mask</i>	The interrupt status flags mask(_sdhc_interrupt_status_flag).

34.7.8 static void SDHC_DisableInterruptSignal (SDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SDHC peripheral base address.
<i>mask</i>	The interrupt status flags mask(_sdhc_interrupt_status_flag).

34.7.9 static uint32_t SDHC_GetEnabledInterruptStatusFlags (SDHC_Type * *base*) [inline], [static]

Parameters

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

Returns

Current interrupt status flags mask(_sdhc_interrupt_status_flag).

34.7.10 static uint32_t SDHC_GetInterruptStatusFlags (SDHC_Type * *base*) [inline], [static]

Parameters

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

Returns

Current interrupt status flags mask(_sdhc_interrupt_status_flag).

34.7.11 static void SDHC_ClearInterruptStatusFlags (SDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SDHC peripheral base address.
<i>mask</i>	The interrupt status flags mask(_sdhc_interrupt_status_flag).

34.7.12 static uint32_t SDHC_GetAutoCommand12ErrorStatusFlags (SDHC_Type * *base*) [inline], [static]

Function Documentation

Parameters

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

Returns

Auto command 12 error status flags mask(_sdhc_auto_command12_error_status_flag).

34.7.13 `static uint32_t SDHC_GetAdmaErrorStatusFlags (SDHC_Type * base)
[inline], [static]`

Parameters

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

Returns

ADMA error status flags mask(_sdhc_adma_error_status_flag).

34.7.14 `static uint32_t SDHC_GetPresentStatusFlags (SDHC_Type * base)
[inline], [static]`

This function gets the present SDHC's status except for an interrupt status and an error status.

Parameters

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

Returns

Present SDHC's status flags mask(_sdhc_present_status_flag).

34.7.15 `void SDHC_GetCapability (SDHC_Type * base, sdhc_capability_t *
capability)`

Parameters

<i>base</i>	SDHC peripheral base address.
<i>capability</i>	Structure to save capability information.

34.7.16 static void SDHC_EnableSdClock (SDHC_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	SDHC peripheral base address.
<i>enable</i>	True to enable, false to disable.

34.7.17 uint32_t SDHC_SetSdClock (SDHC_Type * *base*, uint32_t *srcClock_Hz*, uint32_t *busClock_Hz*)

Parameters

<i>base</i>	SDHC peripheral base address.
<i>srcClock_Hz</i>	SDHC source clock frequency united in Hz.
<i>busClock_Hz</i>	SD bus clock frequency united in Hz.

Returns

The nearest frequency of *busClock_Hz* configured to SD bus.

34.7.18 bool SDHC_SetCardActive (SDHC_Type * *base*, uint32_t *timeout*)

This function must be called each time the card is inserted to ensure that the card can receive the command correctly.

Parameters

Function Documentation

<i>base</i>	SDHC peripheral base address.
<i>timeout</i>	Timeout to initialize card.

Return values

<i>true</i>	Set card active successfully.
<i>false</i>	Set card active failed.

34.7.19 static void SDHC_SetDataBusWidth (SDHC_Type * *base*, sdhc_data_bus_width_t *width*) [inline], [static]

Parameters

<i>base</i>	SDHC peripheral base address.
<i>width</i>	Data transfer width.

34.7.20 static void SDHC_CardDetectByData3 (SDHC_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	SDHC peripheral base address.
<i>enable</i>	Enable/disable flag.

34.7.21 void SDHC_SetTransferConfig (SDHC_Type * *base*, const sdhc_transfer_config_t * *config*)

This function fills the card transfer-related command argument/transfer flag/data size. The command and data are sent by SDHC after calling this function.

Example:

```
sdhc_transfer_config_t transferConfig;  
transferConfig.dataBlockSize = 512U;  
transferConfig.dataBlockCount = 2U;  
transferConfig.commandArgument = 0x01AAU;  
transferConfig.commandIndex = 8U;  
transferConfig.flags |= (kSDHC_EnableDmaFlag |  
    kSDHC_EnableAutoCommand12Flag |  
    kSDHC_MultipleBlockFlag);  
SDHC_SetTransferConfig(SDHC, &transferConfig);
```

Parameters

<i>base</i>	SDHC peripheral base address.
<i>config</i>	Command configuration structure.

34.7.22 static uint32_t SDHC_GetCommandResponse (SDHC_Type * *base*, uint32_t *index*) [inline], [static]

Parameters

<i>base</i>	SDHC peripheral base address.
<i>index</i>	The index of response register, range from 0 to 3.

Returns

Response register transfer.

34.7.23 static void SDHC_WriteData (SDHC_Type * *base*, uint32_t *data*) [inline], [static]

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

<i>base</i>	SDHC peripheral base address.
<i>data</i>	The data about to be sent.

34.7.24 static uint32_t SDHC_ReadData (SDHC_Type * *base*) [inline], [static]

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

Function Documentation

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

Returns

The data has been read.

34.7.25 `static void SDHC_EnableWakeupEvent (SDHC_Type * base, uint32_t mask, bool enable) [inline], [static]`

Parameters

<i>base</i>	SDHC peripheral base address.
<i>mask</i>	Wakeup events mask(_sdhc_wakeup_event).
<i>enable</i>	True to enable, false to disable.

34.7.26 `static void SDHC_EnableCardDetectTest (SDHC_Type * base, bool enable) [inline], [static]`

Parameters

<i>base</i>	SDHC peripheral base address.
<i>enable</i>	True to enable, false to disable.

34.7.27 `static void SDHC_SetCardDetectTestLevel (SDHC_Type * base, bool high) [inline], [static]`

This function sets the card detection test level to indicate whether the card is inserted into the SDHC when DAT[3]/ CD pin is selected as a card detection pin. This function can also assert the pin logic when DAT[3]/CD pin is selected as the card detection pin.

Parameters

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

<i>high</i>	True to set the card detect level to high.
-------------	--

34.7.28 void SDHC_EnableSdioControl (SDHC_Type * *base*, uint32_t *mask*, bool *enable*)

Parameters

<i>base</i>	SDHC peripheral base address.
<i>mask</i>	SDIO card control flags mask(_sdhc_sdio_control_flag).
<i>enable</i>	True to enable, false to disable.

34.7.29 static void SDHC_SetContinueRequest (SDHC_Type * *base*) [inline], [static]

Parameters

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

34.7.30 void SDHC_SetMmcBootConfig (SDHC_Type * *base*, const sdhc_boot_config_t * *config*)

Example:

```
sdhc_boot_config_t config;
config.ackTimeoutCount = 4;
config.bootMode = kSDHC_BootModeNormal;
config.blockCount = 5;
config.enableBootAck = true;
config.enableBoot = true;
config.enableAutoStopAtBlockGap = true;
SDHC_SetMmcBootConfig(SDHC, &config);
```

Parameters

Function Documentation

<i>base</i>	SDHC peripheral base address.
<i>config</i>	The MMC boot configuration information.

34.7.31 static void SDHC_SetForceEvent (SDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SDHC peripheral base address.
<i>mask</i>	The force events mask(_sdhc_force_event).

34.7.32 status_t SDHC_TransferBlocking (SDHC_Type * *base*, uint32_t * *admaTable*, uint32_t *admaTableWords*, sdhc_transfer_t * *transfer*)

This function waits until the command response/data is received or the SDHC encounters an error by polling the status flag. This function support non word align data addr transfer support, if data buffer addr is not align in DMA mode, the API will continue finish the transfer by polling IO directly The application must not call this API in multiple threads at the same time. Because of that this API doesn't support the re-entry mechanism.

Note

There is no need to call the API 'SDHC_TransferCreateHandle' when calling this API.

Parameters

<i>base</i>	SDHC peripheral base address.
<i>admaTable</i>	ADMA table address, can't be null if transfer way is ADMA1/ADMA2.
<i>admaTable- Words</i>	ADMA table length united as words, can't be 0 if transfer way is ADMA1/ADMA2.
<i>transfer</i>	Transfer content.

Return values

<i>kStatus_InvalidArgument</i>	Argument is invalid.
<i>kStatus_SDHC_Prepare-AdmaDescriptorFailed</i>	Prepare ADMA descriptor failed.
<i>kStatus_SDHC_Send-CommandFailed</i>	Send command failed.
<i>kStatus_SDHC_Transfer-DataFailed</i>	Transfer data failed.
<i>kStatus_Success</i>	Operate successfully.

34.7.33 void SDHC_TransferCreateHandle (SDHC_Type * *base*, sdhc_handle_t * *handle*, const sdhc_transfer_callback_t * *callback*, void * *userData*)

Parameters

<i>base</i>	SDHC peripheral base address.
<i>handle</i>	SDHC handle pointer.
<i>callback</i>	Structure pointer to contain all callback functions.
<i>userData</i>	Callback function parameter.

34.7.34 status_t SDHC_TransferNonBlocking (SDHC_Type * *base*, sdhc_handle_t * *handle*, uint32_t * *admaTable*, uint32_t *admaTableWords*, sdhc_transfer_t * *transfer*)

This function sends a command and data and returns immediately. It doesn't wait the transfer complete or encounter an error. This function support non word align data addr transfer support, if data buffer addr is not align in DMA mode, the API will continue finish the transfer by polling IO directly The application must not call this API in multiple threads at the same time. Because of that this API doesn't support the re-entry mechanism.

Note

Call the API 'SDHC_TransferCreateHandle' when calling this API.

Function Documentation

Parameters

<i>base</i>	SDHC peripheral base address.
<i>handle</i>	SDHC handle.
<i>admaTable</i>	ADMA table address, can't be null if transfer way is ADMA1/ADMA2.
<i>admaTable- Words</i>	ADMA table length united as words, can't be 0 if transfer way is ADMA1/ADMA2.
<i>transfer</i>	Transfer content.

Return values

<i>kStatus_InvalidArgument</i>	Argument is invalid.
<i>kStatus_SDHC_Busy- Transferring</i>	Busy transferring.
<i>kStatus_SDHC_Prep- AdmaDescriptorFailed</i>	Prepare ADMA descriptor failed.
<i>kStatus_Success</i>	Operate successfully.

34.7.35 void SDHC_TransferHandleIRQ (SDHC_Type * *base*, sdhc_handle_t * *handle*)

This function deals with the IRQs on the given host controller.

Parameters

<i>base</i>	SDHC peripheral base address.
<i>handle</i>	SDHC handle.

Chapter 35

SIM: System Integration Module Driver

Overview

The MCUXpresso SDK provides a peripheral driver for the System Integration Module (SIM) of MCU-Xpresso 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, 1))

Data Structure Documentation

35.2.1 struct sim_uid_t

Data Fields

- uint32_t [MH](#)

Function Documentation

- *UIDMH.*
uint32_t [ML](#)
- *UIDML.*
uint32_t [L](#)
- *UIDL.*

35.2.1.0.0.31 Field Documentation

35.2.1.0.0.31.1 uint32_t sim_uid_t::MH

35.2.1.0.0.31.2 uint32_t sim_uid_t::ML

35.2.1.0.0.31.3 uint32_t sim_uid_t::L

Enumeration Type Documentation

35.3.1 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.

35.3.2 enum _sim_flash_mode

Enumerator

kSIM_FlashDisableInWait Disable flash in wait mode.

kSIM_FlashDisable Disable flash in normal mode.

Function Documentation

35.4.1 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.
-------------	---------------------------------------

35.4.2 void SIM_GetUniqueld (sim_uid_t * *uid*)

Parameters

<i>uid</i>	Pointer to the structure to save the UID value.
------------	---

35.4.3 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 36

SMC: System Mode Controller Driver

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.

Typical use case

36.2.1 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_smc.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

Typical use case

rw data of fsl_smc.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_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_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` }
 Power Modes in PMSTAT.
- enum `smc_run_mode_t` {
 `kSMC_RunNormal` = `0U`,
 `kSMC_RunVlpr` = `2U` }
 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)
Recovers after wake up from stop modes.
- void [SMC_PreEnterWaitModes](#) (void)
Prepares to enter wait modes.
- void [SMC_PostExitWaitModes](#) (void)
Recovers after wake up from stop modes.
- [status_t](#) [SMC_SetPowerModeRun](#) (SMC_Type *base)
Configures the system to RUN 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, bool wakeupMode)
Configures the system to VLPR power mode.
- [status_t](#) [SMC_SetPowerModeVlprw](#) (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)
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.

Data Structure Documentation

36.3.1 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.

Enumeration Type Documentation

36.4.1 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_AllowPowerModeAll Allow all power mode.

36.4.2 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

36.4.3 enum smc_run_mode_t

Enumerator

kSMC_RunNormal Normal RUN mode.
kSMC_RunVlpr Very-low-power RUN mode.

36.4.4 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.

36.4.5 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.

36.4.6 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.

36.4.7 anonymous enum

Enumerator

kStatus_SMC_StopAbort Entering Stop mode is abort.

Function Documentation

36.5.1 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

Function Documentation

<i>base</i>	SMC peripheral base address.
<i>allowedModes</i>	Bitmap of the allowed power modes.

36.5.2 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.

36.5.3 void SMC_PreEnterStopModes (void)

This function should be called before entering STOP/VLPS/LLS/VLLS modes.

36.5.4 void SMC_PostExitStopModes (void)

This function should be called after wake up from STOP/VLPS/LLS/VLLS modes. It is used with [SMC_PreEnterStopModes](#).

36.5.5 void SMC_PreEnterWaitModes (void)

This function should be called before entering WAIT/VLPW modes.

36.5.6 void SMC_PostExitWaitModes (void)

This function should be called after wake up from WAIT/VLPW modes. It is used with [SMC_PreEnterWaitModes](#).

36.5.7 status_t SMC_SetPowerModeRun (SMC_Type * *base*)

Function Documentation

Parameters

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

Returns

SMC configuration error code.

36.5.8 **status_t SMC_SetPowerModeWait (SMC_Type * *base*)**

Parameters

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

Returns

SMC configuration error code.

36.5.9 **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.

36.5.10 **status_t SMC_SetPowerModeVlpr (SMC_Type * *base*, bool *wakeupMode*)**

Parameters

<i>base</i>	SMC peripheral base address.
<i>wakeupMode</i>	Enter Normal Run mode if true, else stay in VLPR mode.

Returns

SMC configuration error code.

36.5.11 **status_t SMC_SetPowerModeVlpw (SMC_Type * *base*)**

Parameters

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

Returns

SMC configuration error code.

36.5.12 **status_t SMC_SetPowerModeVlps (SMC_Type * *base*)**

Parameters

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

Returns

SMC configuration error code.

36.5.13 **status_t SMC_SetPowerModeLls (SMC_Type * *base*)**

Parameters

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

Returns

SMC configuration error code.

36.5.14 **status_t SMC_SetPowerModeVlls (SMC_Type * *base*, const smc_power_mode_vlls_config_t * *config*)**

Function Documentation

Parameters

<i>base</i>	SMC peripheral base address.
<i>config</i>	The VLLS power mode configuration structure.

Returns

SMC configuration error code.

Chapter 37

SYSMPU: System Memory Protection Unit

Overview

The SYSMPU driver provides hardware access control for all memory references generated in the device. Use the SYSMPU driver to program the region descriptors that define memory spaces and their access rights. After initialization, the SYSMPU concurrently monitors the system bus transactions and evaluates their appropriateness.

Initialization and Deinitialization

To initialize the SYSMPU module, call the [SYSMPU_Init\(\)](#) function and provide the user configuration data structure. This function sets the configuration of the SYSMPU module automatically and enables the SYSMPU module.

Note that the configuration start address, end address, the region valid value, and the debugger's access permission for the SYSMPU region 0 cannot be changed.

This is an example code to configure the SYSMPU driver.

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

Basic Control Operations

SYSMPU can be enabled/disabled for the entire memory protection region by calling the [SYSMPU_Enable\(\)](#) function. To save the power for any unused special regions when the entire memory protection region is disabled, call the [SYSMPU_RegionEnable\(\)](#).

After SYSMPU initialization, the [SYSMPU_SetRegionLowMasterAccessRights\(\)](#) and [SYSMPU_SetRegionHighMasterAccessRights\(\)](#) can be used to change the access rights for special master ports and for special region numbers. The [SYSMPU_SetRegionConfig](#) can be used to set the whole region with the start/end address with access rights.

The [SYSMPU_GetHardwareInfo\(\)](#) API is provided to get the hardware information for the device. The [SYSMPU_GetSlavePortErrorStatus\(\)](#) API is provided to get the error status of a special slave port. When an error happens in this port, the [SYSMPU_GetDetailErrorAccessInfo\(\)](#) API is provided to get the detailed error information.

Data Structures

- struct [sysmpu_hardware_info_t](#)
SYSMPU hardware basic information. [More...](#)
- struct [sysmpu_access_err_info_t](#)
SYSMPU detail error access information. [More...](#)
- struct [sysmpu_rwxrights_master_access_control_t](#)

Basic Control Operations

- *SYSMPU read/write/execute rights control for bus master 0 ~ 3. [More...](#)*
- struct [sysmpu_rwrights_master_access_control_t](#)
SYSMPU read/write access control for bus master 4 ~ 7. [More...](#)
- struct [sysmpu_region_config_t](#)
SYSMPU region configuration structure. [More...](#)
- struct [sysmpu_config_t](#)
The configuration structure for the SYSMPU initialization. [More...](#)

Macros

- #define [SYSMPU_MASTER_RWATTRIBUTE_START_PORT](#) (4U)
define the start master port with read and write attributes.
- #define [SYSMPU_REGION_RWXRIGHTS_MASTER_SHIFT](#)(n) ((n)*6U)
SYSMPU the bit shift for masters with privilege rights: read write and execute.
- #define [SYSMPU_REGION_RWXRIGHTS_MASTER_MASK](#)(n) (0x1FUL << SYSMPU_REGION_RWXRIGHTS_MASTER_SHIFT(n))
SYSMPU masters with read, write and execute rights bit mask.
- #define [SYSMPU_REGION_RWXRIGHTS_MASTER_WIDTH](#) 5U
SYSMPU masters with read, write and execute rights bit width.
- #define [SYSMPU_REGION_RWXRIGHTS_MASTER](#)(n, x) (((uint32_t)((uint32_t)(x)) << SYSMPU_REGION_RWXRIGHTS_MASTER_SHIFT(n))) & [SYSMPU_REGION_RWXRIGHTS_MASTER_MASK](#)(n)
SYSMPU masters with read, write and execute rights priority setting.
- #define [SYSMPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT](#)(n) ((n)*6U + SYSMPU_REGION_RWXRIGHTS_MASTER_WIDTH)
SYSMPU masters with read, write and execute rights process enable bit shift.
- #define [SYSMPU_REGION_RWXRIGHTS_MASTER_PE_MASK](#)(n) (0x1UL << SYSMPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT(n))
SYSMPU masters with read, write and execute rights process enable bit mask.
- #define [SYSMPU_REGION_RWXRIGHTS_MASTER_PE](#)(n, x)
SYSMPU masters with read, write and execute rights process enable setting.
- #define [SYSMPU_REGION_RWRIGHTS_MASTER_SHIFT](#)(n) (((n)-[SYSMPU_MASTER_RWATTRIBUTE_START_PORT](#)) * 2U + 24U)
SYSMPU masters with normal read write permission bit shift.
- #define [SYSMPU_REGION_RWRIGHTS_MASTER_MASK](#)(n) (0x3UL << SYSMPU_REGION_RWRIGHTS_MASTER_SHIFT(n))
SYSMPU masters with normal read write rights bit mask.
- #define [SYSMPU_REGION_RWRIGHTS_MASTER](#)(n, x) (((uint32_t)((uint32_t)(x)) << SYSMPU_REGION_RWRIGHTS_MASTER_SHIFT(n))) & [SYSMPU_REGION_RWRIGHTS_MASTER_MASK](#)(n)
SYSMPU masters with normal read write rights priority setting.

Enumerations

- enum [sysmpu_region_total_num_t](#) {
 [kSYSMPU_8Regions](#) = 0x0U,
 [kSYSMPU_12Regions](#) = 0x1U,
 [kSYSMPU_16Regions](#) = 0x2U }
Describes the number of SYSMPU regions.

- enum `sysmpu_slave_t` {
`kSYSMPU_Slave0` = 0U,
`kSYSMPU_Slave1` = 1U,
`kSYSMPU_Slave2` = 2U,
`kSYSMPU_Slave3` = 3U,
`kSYSMPU_Slave4` = 4U }
SYSMPU slave port number.
- enum `sysmpu_err_access_control_t` {
`kSYSMPU_NoRegionHit` = 0U,
`kSYSMPU_NoneOverlappRegion` = 1U,
`kSYSMPU_OverlappRegion` = 2U }
SYSMPU error access control detail.
- enum `sysmpu_err_access_type_t` {
`kSYSMPU_ErrTypeRead` = 0U,
`kSYSMPU_ErrTypeWrite` = 1U }
SYSMPU error access type.
- enum `sysmpu_err_attributes_t` {
`kSYSMPU_InstructionAccessInUserMode` = 0U,
`kSYSMPU_DataAccessInUserMode` = 1U,
`kSYSMPU_InstructionAccessInSupervisorMode` = 2U,
`kSYSMPU_DataAccessInSupervisorMode` = 3U }
SYSMPU access error attributes.
- enum `sysmpu_supervisor_access_rights_t` {
`kSYSMPU_SupervisorReadWriteExecute` = 0U,
`kSYSMPU_SupervisorReadExecute` = 1U,
`kSYSMPU_SupervisorReadWrite` = 2U,
`kSYSMPU_SupervisorEqualToUsermode` = 3U }
SYSMPU access rights in supervisor mode for bus master 0 ~ 3.
- enum `sysmpu_user_access_rights_t` {
`kSYSMPU_UserNoAccessRights` = 0U,
`kSYSMPU_UserExecute` = 1U,
`kSYSMPU_UserWrite` = 2U,
`kSYSMPU_UserWriteExecute` = 3U,
`kSYSMPU_UserRead` = 4U,
`kSYSMPU_UserReadExecute` = 5U,
`kSYSMPU_UserReadWrite` = 6U,
`kSYSMPU_UserReadWriteExecute` = 7U }
SYSMPU access rights in user mode for bus master 0 ~ 3.

Driver version

- #define `FSL_SYSMPU_DRIVER_VERSION` (`MAKE_VERSION(2, 2, 3)`)
SYSMPU driver version 2.2.3.

Initialization and deinitialization

- void `SYSMPU_Init` (`SYSMPU_Type *base`, const `sysmpu_config_t *config`)

Data Structure Documentation

- Initializes the SYSMPU with the user configuration structure.*
- void [SYSMPU_Deinit](#) (SYSMPU_Type *base)
Deinitializes the SYSMPU regions.

Basic Control Operations

- static void [SYSMPU_Enable](#) (SYSMPU_Type *base, bool enable)
Enables/disables the SYSMPU globally.
- static void [SYSMPU_RegionEnable](#) (SYSMPU_Type *base, uint32_t number, bool enable)
Enables/disables the SYSMPU for a special region.
- void [SYSMPU_GetHardwareInfo](#) (SYSMPU_Type *base, [sysmpu_hardware_info_t](#) *hardwareInform)
Gets the SYSMPU basic hardware information.
- void [SYSMPU_SetRegionConfig](#) (SYSMPU_Type *base, const [sysmpu_region_config_t](#) *regionConfig)
Sets the SYSMPU region.
- void [SYSMPU_SetRegionAddr](#) (SYSMPU_Type *base, uint32_t regionNum, uint32_t startAddr, uint32_t endAddr)
Sets the region start and end address.
- void [SYSMPU_SetRegionRwxMasterAccessRights](#) (SYSMPU_Type *base, uint32_t regionNum, uint32_t masterNum, const [sysmpu_rwxrights_master_access_control_t](#) *accessRights)
Sets the SYSMPU region access rights for masters with read, write, and execute rights.
- void [SYSMPU_SetRegionRwMasterAccessRights](#) (SYSMPU_Type *base, uint32_t regionNum, uint32_t masterNum, const [sysmpu_rwrights_master_access_control_t](#) *accessRights)
Sets the SYSMPU region access rights for masters with read and write rights.
- bool [SYSMPU_GetSlavePortErrorStatus](#) (SYSMPU_Type *base, [sysmpu_slave_t](#) slaveNum)
Gets the numbers of slave ports where errors occur.
- void [SYSMPU_GetDetailErrorAccessInfo](#) (SYSMPU_Type *base, [sysmpu_slave_t](#) slaveNum, [sysmpu_access_err_info_t](#) *errInform)
Gets the SYSMPU detailed error access information.

Data Structure Documentation

37.4.1 struct [sysmpu_hardware_info_t](#)

Data Fields

- uint8_t [hardwareRevisionLevel](#)
Specifies the SYSMPU's hardware and definition reversion level.
- uint8_t [slavePortsNumbers](#)
Specifies the number of slave ports connected to SYSMPU.
- [sysmpu_region_total_num_t](#) [regionsNumbers](#)
Indicates the number of region descriptors implemented.

37.4.1.0.0.32 Field Documentation**37.4.1.0.0.32.1** `uint8_t sysmpu_hardware_info_t::hardwareRevisionLevel`**37.4.1.0.0.32.2** `uint8_t sysmpu_hardware_info_t::slavePortsNumbers`**37.4.1.0.0.32.3** `sysmpu_region_total_num_t sysmpu_hardware_info_t::regionsNumbers`**37.4.2 struct sysmpu_access_err_info_t****Data Fields**

- `uint32_t master`
Access error master.
- `sysmpu_err_attributes_t attributes`
Access error attributes.
- `sysmpu_err_access_type_t accessType`
Access error type.
- `sysmpu_err_access_control_t accessControl`
Access error control.
- `uint32_t address`
Access error address.
- `uint8_t processorIdentification`
Access error processor identification.

37.4.2.0.0.33 Field Documentation**37.4.2.0.0.33.1** `uint32_t sysmpu_access_err_info_t::master`**37.4.2.0.0.33.2** `sysmpu_err_attributes_t sysmpu_access_err_info_t::attributes`**37.4.2.0.0.33.3** `sysmpu_err_access_type_t sysmpu_access_err_info_t::accessType`**37.4.2.0.0.33.4** `sysmpu_err_access_control_t sysmpu_access_err_info_t::accessControl`**37.4.2.0.0.33.5** `uint32_t sysmpu_access_err_info_t::address`**37.4.2.0.0.33.6** `uint8_t sysmpu_access_err_info_t::processorIdentification`**37.4.3 struct sysmpu_rwxrights_master_access_control_t****Data Fields**

- `sysmpu_supervisor_access_rights_t superAccessRights`
Master access rights in supervisor mode.
- `sysmpu_user_access_rights_t userAccessRights`
Master access rights in user mode.
- `bool processIdentifierEnable`
Enables or disables process identifier.

37.4.3.0.0.34 Field Documentation

37.4.3.0.0.34.1 `sysmpu_supervisor_access_rights_t sysmpu_rwxrights_master_access_control_t::superAccessRights`

37.4.3.0.0.34.2 `sysmpu_user_access_rights_t sysmpu_rwxrights_master_access_control_t::userAccessRights`

37.4.3.0.0.34.3 `bool sysmpu_rwxrights_master_access_control_t::processIdentifierEnable`

37.4.4 `struct sysmpu_rwrights_master_access_control_t`

Data Fields

- `bool writeEnable`
Enables or disables write permission.
- `bool readEnable`
Enables or disables read permission.

37.4.4.0.0.35 Field Documentation

37.4.4.0.0.35.1 `bool sysmpu_rwrights_master_access_control_t::writeEnable`

37.4.4.0.0.35.2 `bool sysmpu_rwrights_master_access_control_t::readEnable`

37.4.5 `struct sysmpu_region_config_t`

This structure is used to configure the `regionNum` region. The `accessRights1[0] ~ accessRights1[3]` are used to configure the bus master 0 ~ 3 with the privilege rights setting. The `accessRights2[0] ~ accessRights2[3]` are used to configure the high master 4 ~ 7 with the normal read write permission. The master port assignment is the chip configuration. Normally, the core is the master 0, debugger is the master 1. Note that the SYSPMU assigns a priority scheme where the debugger is treated as the highest priority master followed by the core and then all the remaining masters. SYSPMU protection does not allow writes from the core to affect the "regionNum 0" start and end address nor the permissions associated with the debugger. It can only write the permission fields associated with the other masters. This protection guarantees that the debugger always has access to the entire address space and those rights can't be changed by the core or any other bus master. Prepare the region configuration when `regionNum` is 0.

Data Fields

- `uint32_t regionNum`
SYSPMU region number, range form 0 ~ `FSL_FEATURE_SYSPMU_DESCRIPTOR_COUNT` - 1.
- `uint32_t startAddress`
Memory region start address.
- `uint32_t endAddress`
Memory region end address.
- `sysmpu_rwxrights_master_access_control_t accessRights1 [4]`

- Masters with read, write and execute rights setting.*
- [sysmpu_rwrights_master_access_control_t accessRights2](#) [4]
Masters with normal read write rights setting.
- [uint8_t processIdentifier](#)
Process identifier used when "processIdentifierEnable" set with true.
- [uint8_t processIdMask](#)
Process identifier mask.

37.4.5.0.0.36 Field Documentation

37.4.5.0.0.36.1 [uint32_t sysmpu_region_config_t::regionNum](#)

37.4.5.0.0.36.2 [uint32_t sysmpu_region_config_t::startAddress](#)

Note: bit0 ~ bit4 always be marked as 0 by SYSMPU. The actual start address is 0-modulo-32 byte address.

37.4.5.0.0.36.3 [uint32_t sysmpu_region_config_t::endAddress](#)

Note: bit0 ~ bit4 always be marked as 1 by SYSMPU. The actual end address is 31-modulo-32 byte address.

37.4.5.0.0.36.4 [sysmpu_rwxrights_master_access_control_t sysmpu_region_config_t::accessRights1](#)[4]

37.4.5.0.0.36.5 [sysmpu_rwrights_master_access_control_t sysmpu_region_config_t::accessRights2](#)[4]

37.4.5.0.0.36.6 [uint8_t sysmpu_region_config_t::processIdentifier](#)

37.4.5.0.0.36.7 [uint8_t sysmpu_region_config_t::processIdMask](#)

The setting bit will ignore the same bit in process identifier.

37.4.6 struct [sysmpu_config_t](#)

This structure is used when calling the SYSMPU_Init function.

Data Fields

- [sysmpu_region_config_t regionConfig](#)
Region access permission.
- [struct _sysmpu_config * next](#)
Pointer to the next structure.

Macro Definition Documentation

37.4.6.0.0.37 Field Documentation

37.4.6.0.0.37.1 `sysmpu_region_config_t sysmpu_config_t::regionConfig`

37.4.6.0.0.37.2 `struct _sysmpu_config* sysmpu_config_t::next`

Macro Definition Documentation

37.5.1 `#define FSL_SYSPMU_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))`

37.5.2 `#define SYSPMU_MASTER_RWATTRIBUTE_START_PORT (4U)`

37.5.3 `#define SYSPMU_REGION_RWXRIGHTS_MASTER_SHIFT(n) ((n)*6U)`

37.5.4 `#define SYSPMU_REGION_RWXRIGHTS_MASTER_MASK(n) (0x1FUL << SYSPMU_REGION_RWXRIGHTS_MASTER_SHIFT(n))`

37.5.5 `#define SYSPMU_REGION_RWXRIGHTS_MASTER_WIDTH 5U`

37.5.6 `#define SYSPMU_REGION_RWXRIGHTS_MASTER(n, x) (((uint32_t)((uint32_t)(x)) << SYSPMU_REGION_RWXRIGHTS_MASTER_SHIFT(n))) & SYSPMU_REGION_RWXRIGHTS_MASTER_MASK(n))`

37.5.7 `#define SYSPMU_REGION_RWXRIGHTS_MASTER_PE_SHIFT(n) ((n)*6U + SYSPMU_REGION_RWXRIGHTS_MASTER_WIDTH)`

37.5.8 `#define SYSPMU_REGION_RWXRIGHTS_MASTER_PE_MASK(n) (0x1UL << SYSPMU_REGION_RWXRIGHTS_MASTER_PE_SHIFT(n))`

37.5.9 `#define SYSPMU_REGION_RWXRIGHTS_MASTER_PE(n, x)`

Value:

```
((uint32_t)((uint32_t)(x)) << SYSPMU_REGION_RWXRIGHTS_MASTER_PE_SHIFT(n))) & \
SYSPMU_REGION_RWXRIGHTS_MASTER_PE_MASK(n))
```

37.5.10 `#define SYSPMU_REGION_RWRIGHTS_MASTER_SHIFT(n) (((n)-SYSPMU_MASTER_RWATTRIBUTE_START_PORT) * 2U + 24U)`

37.5.11 `#define SYSPMU_REGION_RWRIGHTS_MASTER_MASK(n) (0x3UL << SYSPMU_REGION_RWRIGHTS_MASTER_SHIFT(n))`

37.5.12 `#define SYSPMU_REGION_RWRIGHTS_MASTER(n, x) (((uint32_t)((uint32_t)(x)) << SYSPMU_REGION_RWRIGHTS_MASTER_SHIFT(n))) & SYSPMU_REGION_RWRIGHTS_MASTER_MASK(n))`

Enumeration Type Documentation

37.6.1 `enum sysmpu_region_total_num_t`

Enumerator

kSYSPMU_8Regions SYSPMU supports 8 regions.
kSYSPMU_12Regions SYSPMU supports 12 regions.
kSYSPMU_16Regions SYSPMU supports 16 regions.

37.6.2 `enum sysmpu_slave_t`

Enumerator

kSYSPMU_Slave0 SYSPMU slave port 0.
kSYSPMU_Slave1 SYSPMU slave port 1.
kSYSPMU_Slave2 SYSPMU slave port 2.
kSYSPMU_Slave3 SYSPMU slave port 3.
kSYSPMU_Slave4 SYSPMU slave port 4.

37.6.3 `enum sysmpu_err_access_control_t`

Enumerator

kSYSPMU_NoRegionHit No region hit error.
kSYSPMU_NoneOverlappRegion Access single region error.
kSYSPMU_OverlappRegion Access overlapping region error.

Enumeration Type Documentation

37.6.4 enum sysmpu_err_access_type_t

Enumerator

kSYSMPU_ErrTypeRead SYSMPU error access type — read.

kSYSMPU_ErrTypeWrite SYSMPU error access type — write.

37.6.5 enum sysmpu_err_attributes_t

Enumerator

kSYSMPU_InstructionAccessInUserMode Access instruction error in user mode.

kSYSMPU_DataAccessInUserMode Access data error in user mode.

kSYSMPU_InstructionAccessInSupervisorMode Access instruction error in supervisor mode.

kSYSMPU_DataAccessInSupervisorMode Access data error in supervisor mode.

37.6.6 enum sysmpu_supervisor_access_rights_t

Enumerator

kSYSMPU_SupervisorReadWriteExecute Read write and execute operations are allowed in supervisor mode.

kSYSMPU_SupervisorReadExecute Read and execute operations are allowed in supervisor mode.

kSYSMPU_SupervisorReadWrite Read write operations are allowed in supervisor mode.

kSYSMPU_SupervisorEqualToUsermode Access permission equal to user mode.

37.6.7 enum sysmpu_user_access_rights_t

Enumerator

kSYSMPU_UserNoAccessRights No access allowed in user mode.

kSYSMPU_UserExecute Execute operation is allowed in user mode.

kSYSMPU_UserWrite Write operation is allowed in user mode.

kSYSMPU_UserWriteExecute Write and execute operations are allowed in user mode.

kSYSMPU_UserRead Read is allowed in user mode.

kSYSMPU_UserReadExecute Read and execute operations are allowed in user mode.

kSYSMPU_UserReadWrite Read and write operations are allowed in user mode.

kSYSMPU_UserReadWriteExecute Read write and execute operations are allowed in user mode.

Function Documentation

37.7.1 void SYSMPU_Init (SYSMPU_Type * *base*, const sysmpu_config_t * *config*)

This function configures the SYSMPU module with the user-defined configuration.

Function Documentation

Parameters

<i>base</i>	SYSMPU peripheral base address.
<i>config</i>	The pointer to the configuration structure.

37.7.2 void SYSMPU_Deinit (SYSMPU_Type * *base*)

Parameters

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

37.7.3 static void SYSMPU_Enable (SYSMPU_Type * *base*, bool *enable*) [inline], [static]

Call this API to enable or disable the SYSMPU module.

Parameters

<i>base</i>	SYSMPU peripheral base address.
<i>enable</i>	True enable SYSMPU, false disable SYSMPU.

37.7.4 static void SYSMPU_RegionEnable (SYSMPU_Type * *base*, uint32_t *number*, bool *enable*) [inline], [static]

When SYSMPU is enabled, call this API to disable an unused region of an enabled SYSMPU. Call this API to minimize the power dissipation.

Parameters

<i>base</i>	SYSMPU peripheral base address.
<i>number</i>	SYSMPU region number.
<i>enable</i>	True enable the special region SYSMPU, false disable the special region SYSMPU.

37.7.5 void SYSMPU_GetHardwareInfo (SYSMPU_Type * *base*, sysmpu_hardware_info_t * *hardwareInform*)

Parameters

<i>base</i>	SYSMPU peripheral base address.
<i>hardware- Inform</i>	The pointer to the SYSMPU hardware information structure. See "sysmpu_hardware_info_t".

37.7.6 void SYSMPU_SetRegionConfig (SYSMPU_Type * *base*, const sysmpu_region_config_t * *regionConfig*)

Note: Due to the SYSMPU protection, the region number 0 does not allow writes from core to affect the start and end address nor the permissions associated with the debugger. It can only write the permission fields associated with the other masters.

Parameters

<i>base</i>	SYSMPU peripheral base address.
<i>regionConfig</i>	The pointer to the SYSMPU user configuration structure. See "sysmpu_region_config_t".

37.7.7 void SYSMPU_SetRegionAddr (SYSMPU_Type * *base*, uint32_t *regionNum*, uint32_t *startAddr*, uint32_t *endAddr*)

Memory region start address. Note: bit0 ~ bit4 is always marked as 0 by SYSMPU. The actual start address by SYSMPU is 0-modulo-32 byte address. Memory region end address. Note: bit0 ~ bit4 always be marked as 1 by SYSMPU. The end address used by the SYSMPU is 31-modulo-32 byte address. Note: Due to the SYSMPU protection, the startAddr and endAddr can't be changed by the core when regionNum is 0.

Parameters

<i>base</i>	SYSMPU peripheral base address.
<i>regionNum</i>	SYSMPU region number. The range is from 0 to FSL_FEATURE_SYSMPU_DESCRIPTOR_COUNT - 1.
<i>startAddr</i>	Region start address.
<i>endAddr</i>	Region end address.

37.7.8 void SYSPMU_SetRegionRwxMasterAccessRights (SYSPMU_Type * *base*, uint32_t *regionNum*, uint32_t *masterNum*, const sysmpu_rwxrights_master_access_control_t * *accessRights*)

The SYSPMU access rights depend on two board classifications of bus masters. The privilege rights masters and the normal rights masters. The privilege rights masters have the read, write, and execute access rights. Except the normal read and write rights, the execute rights are also allowed for these masters. The privilege rights masters normally range from bus masters 0 - 3. However, the maximum master number is device-specific. See the "SYSPMU_PRIVILEGED_RIGHTS_MASTER_MAX_INDEX". The normal rights masters access rights control see "SYSPMU_SetRegionRwMasterAccessRights()".

Parameters

<i>base</i>	SYSPMU peripheral base address.
<i>regionNum</i>	SYSPMU region number. Should range from 0 to FSL_FEATURE_SYSPMU_DESCRIPTOR_COUNT - 1.
<i>masterNum</i>	SYSPMU bus master number. Should range from 0 to SYSPMU_PRIVILEGED_RIGHTS_MASTER_MAX_INDEX.
<i>accessRights</i>	The pointer to the SYSPMU access rights configuration. See "sysmpu_rwxrights_master_access_control_t".

37.7.9 void SYSPMU_SetRegionRwMasterAccessRights (SYSPMU_Type * *base*, uint32_t *regionNum*, uint32_t *masterNum*, const sysmpu_rwrights_master_access_control_t * *accessRights*)

The SYSPMU access rights depend on two board classifications of bus masters. The privilege rights masters and the normal rights masters. The normal rights masters only have the read and write access permissions. The privilege rights access control see "SYSPMU_SetRegionRwxMasterAccessRights".

Parameters

<i>base</i>	SYSPMU peripheral base address.
<i>regionNum</i>	SYSPMU region number. The range is from 0 to FSL_FEATURE_SYSPMU_DESCRIPTOR_COUNT - 1.
<i>masterNum</i>	SYSPMU bus master number. Should range from SYSPMU_MASTER_RWATTRIBUTE_START_PORT to ~ FSL_FEATURE_SYSPMU_MASTER_COUNT - 1.

<i>accessRights</i>	The pointer to the SYSMPU access rights configuration. See "sysmpu_rwrights_master_access_control_t".
---------------------	---

37.7.10 **bool SYSMPU_GetSlavePortErrorStatus (SYSMPU_Type * *base*, sysmpu_slave_t *slaveNum*)**

Parameters

<i>base</i>	SYSMPU peripheral base address.
<i>slaveNum</i>	SYSMPU slave port number.

Returns

The slave ports error status. true - error happens in this slave port. false - error didn't happen in this slave port.

37.7.11 **void SYSMPU_GetDetailErrorAccessInfo (SYSMPU_Type * *base*, sysmpu_slave_t *slaveNum*, sysmpu_access_err_info_t * *errInform*)**

Parameters

<i>base</i>	SYSMPU peripheral base address.
<i>slaveNum</i>	SYSMPU slave port number.
<i>errInform</i>	The pointer to the SYSMPU access error information. See "sysmpu_access_err_info_t".



Chapter 38

UART: Universal Asynchronous Receiver/Transmitter Driver

Overview

Modules

- [UART CMSIS Driver](#)
- [UART Driver](#)
- [UART FreeRTOS Driver](#)
- [UART eDMA Driver](#)

UART Driver

38.2.1 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_RxRingBufferOverflow`. 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.

38.2.2 Typical use case

38.2.2.1 UART Send/receive using a polling method

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

38.2.2.2 UART Send/receive using an interrupt method

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

38.2.2.3 UART Receive using the ringbuffer feature

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

38.2.2.4 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 `_uart_flags` {


```

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.

Driver version

- #define `FSL_UART_DRIVER_VERSION` (MAKE_VERSION(2, 3, 0))
UART driver version 2.3.0.

Initialization and deinitialization

- status_t `UART_Init` (UART_Type *base, const `uart_config_t` *config, uint32_t srcClock_Hz)
Initializes a UART instance with a user configuration structure and peripheral clock.
- 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.

UART Driver

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.
- 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.
- void [UART_TransferHandleIRQ](#) (UART_Type *base, uart_handle_t *handle)
UART IRQ handle function.
- void [UART_TransferHandleErrorIRQ](#) (UART_Type *base, uart_handle_t *handle)
UART Error IRQ handle function.

38.2.3 Data Structure Documentation

38.2.3.1 struct uart_config_t

Data Fields

- uint32_t [baudRate_Bps](#)
UART baud rate.
- [uart_parity_mode_t](#) [parityMode](#)
Parity mode, disabled (default), even, odd.
- [uart_stop_bit_count_t](#) [stopBitCount](#)
Number of stop bits, 1 stop bit (default) or 2 stop bits.
- uint8_t [txFifoWatermark](#)
TX FIFO watermark.
- uint8_t [rxFifoWatermark](#)
RX FIFO watermark.
- bool [enableRxRTS](#)

UART Driver

- *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.

38.2.3.1.0.38 Field Documentation

38.2.3.1.0.38.1 [uart_idle_type_select_t](#) [uart_config_t::idleType](#)

38.2.3.2 struct [uart_transfer_t](#)

Data Fields

- [uint8_t](#) * [data](#)
The buffer of data to be transfer.
- [size_t](#) [dataSize](#)
The byte count to be transfer.

38.2.3.2.0.39 Field Documentation

38.2.3.2.0.39.1 [uint8_t*](#) [uart_transfer_t::data](#)

38.2.3.2.0.39.2 [size_t](#) [uart_transfer_t::dataSize](#)

38.2.3.3 struct [_uart_handle](#)

Data Fields

- [uint8_t](#) *volatile [txData](#)
Address of remaining data to send.
- volatile [size_t](#) [txDataSize](#)
Size of the remaining data to send.
- [size_t](#) [txDataSizeAll](#)
Size of the data to send out.
- [uint8_t](#) *volatile [rxData](#)
Address of remaining data to receive.
- volatile [size_t](#) [rxDataSize](#)
Size of the remaining data to receive.
- [size_t](#) [rxDataSizeAll](#)
Size of the data to receive.
- [uint8_t](#) * [rxRingBuffer](#)
Start address of the receiver ring buffer.
- [size_t](#) [rxRingBufferSize](#)
Size of the ring buffer.
- volatile [uint16_t](#) [rxRingBufferHead](#)
Index for the driver to store received data into ring buffer.

- volatile uint16_t [rxRingBufferTail](#)
Index for the user to get data from the ring buffer.
- [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.

UART Driver

38.2.3.3.0.40 Field Documentation

- 38.2.3.3.0.40.1 `uint8_t* volatile uart_handle_t::txData`
- 38.2.3.3.0.40.2 `volatile size_t uart_handle_t::txDataSize`
- 38.2.3.3.0.40.3 `size_t uart_handle_t::txDataSizeAll`
- 38.2.3.3.0.40.4 `uint8_t* volatile uart_handle_t::rxData`
- 38.2.3.3.0.40.5 `volatile size_t uart_handle_t::rxDataSize`
- 38.2.3.3.0.40.6 `size_t uart_handle_t::rxDataSizeAll`
- 38.2.3.3.0.40.7 `uint8_t* uart_handle_t::rxRingBuffer`
- 38.2.3.3.0.40.8 `size_t uart_handle_t::rxRingBufferSize`
- 38.2.3.3.0.40.9 `volatile uint16_t uart_handle_t::rxRingBufferHead`
- 38.2.3.3.0.40.10 `volatile uint16_t uart_handle_t::rxRingBufferTail`
- 38.2.3.3.0.40.11 `uart_transfer_callback_t uart_handle_t::callback`
- 38.2.3.3.0.40.12 `void* uart_handle_t::userData`
- 38.2.3.3.0.40.13 `volatile uint8_t uart_handle_t::txState`

38.2.4 Macro Definition Documentation

- 38.2.4.1 `#define FSL_UART_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))`
- 38.2.4.2 `#define UART_RETRY_TIMES 0U /* Defining to zero means to keep waiting for the flag until it is assert/deassert. */`

38.2.5 Typedef Documentation

- 38.2.5.1 `typedef void(* uart_transfer_callback_t)(UART_Type *base, uart_handle_t *handle, status_t status, void *userData)`

38.2.6 Enumeration Type Documentation

38.2.6.1 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_RxRingBufferOverflow UART RX software ring buffer overrun.
kStatus_UART_RxHardwareOverflow 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.

38.2.6.2 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.

38.2.6.3 enum uart_stop_bit_count_t

Enumerator

kUART_OneStopBit One stop bit.
kUART_TwoStopBit Two stop bits.

38.2.6.4 enum uart_idle_type_select_t

Enumerator

kUART_IdleTypeStartBit Start counting after a valid start bit.
kUART_IdleTypeStopBit Start counting after a stop bit.

38.2.6.5 enum _uart_interrupt_enable

This structure contains the settings for all of the UART interrupt configurations.

UART Driver

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.

38.2.6.6 enum _uart_flags

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.

38.2.7 Function Documentation

38.2.7.1 `uint32_t UART_GetInstance (UART_Type * base)`

UART Driver

Parameters

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

Returns

UART instance.

38.2.7.2 `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

38.2.7.3 `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.
-------------	-------------------------------

38.2.7.4 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.
---------------	-------------------------------------

38.2.7.5 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, 200000000U);
*
```

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.
---	--

UART Driver

<i>kStatus_Success</i>	Set baudrate succeeded.
------------------------	-------------------------

38.2.7.6 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](#).

38.2.7.7 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 _uart_flags .

Return values

<i>kStatus_UART_Flag- CannotClearManually</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.

38.2.7.8 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 .

38.2.7.9 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 .

38.2.7.10 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.

UART Driver

```
*      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](#).

38.2.7.11 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.

38.2.7.12 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.

38.2.7.13 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.

38.2.7.14 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.

38.2.7.15 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.

38.2.7.16 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.

38.2.7.17 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.

UART Driver

Parameters

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

Returns

The byte read from UART data register.

38.2.7.18 **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.

38.2.7.19 **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.

38.2.7.20 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.

38.2.7.21 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.

UART Driver

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.

38.2.7.22 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.

38.2.7.23 size_t UART_TransferGetRxRingBufferLength (uart_handle_t * *handle*)

Parameters

<i>handle</i>	UART handle pointer.
---------------	----------------------

Returns

Length of received data in RX ring buffer.

38.2.7.24 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.

38.2.7.25 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.

38.2.7.26 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

UART Driver

<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 count;

38.2.7.27 **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 [kStatus_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.

38.2.7.28 **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.

38.2.7.29 **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> ;

38.2.7.30 **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.

38.2.7.31 **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.

38.2.7.32 void UART_TransferHandleIRQ (UART_Type * *base*, uart_handle_t * *handle*)

This function handles the UART transmit and receive IRQ request.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.

38.2.7.33 void UART_TransferHandleErrorIRQ (UART_Type * *base*, uart_handle_t * *handle*)

This function handles the UART error IRQ request.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.

UART eDMA Driver

UART eDMA Driver

38.3.1 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, 3, 0))
UART EDMA driver version 2.3.0.

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.

38.3.2 Data Structure Documentation

38.3.2.1 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.

UART eDMA Driver

38.3.2.1.0.41 Field Documentation

38.3.2.1.0.41.1 `uart_edma_transfer_callback_t` `uart_edma_handle_t::callback`

38.3.2.1.0.41.2 `void*` `uart_edma_handle_t::userData`

38.3.2.1.0.41.3 `size_t` `uart_edma_handle_t::rxDataSizeAll`

38.3.2.1.0.41.4 `size_t` `uart_edma_handle_t::txDataSizeAll`

38.3.2.1.0.41.5 `edma_handle_t*` `uart_edma_handle_t::txEdmaHandle`

38.3.2.1.0.41.6 `edma_handle_t*` `uart_edma_handle_t::rxEdmaHandle`

38.3.2.1.0.41.7 `uint8_t` `uart_edma_handle_t::nbytes`

38.3.2.1.0.41.8 `volatile uint8_t` `uart_edma_handle_t::txState`

38.3.3 Macro Definition Documentation

38.3.3.1 `#define FSL_UART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))`

38.3.4 Typedef Documentation

38.3.4.1 `typedef void(* uart_edma_transfer_callback_t)(UART_Type *base,
uart_edma_handle_t *handle, status_t status, void *userData)`

38.3.5 Function Documentation

38.3.5.1 `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.

38.3.5.2 `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.

38.3.5.3 `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

UART eDMA Driver

<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.

38.3.5.4 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.

38.3.5.5 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.

38.3.5.6 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_NoTransferInProgress</i>	No send in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <code>count</code> ;

38.3.5.7 **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_NoTransferInProgress</i>	No receive in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <code>count</code> ;

UART FreeRTOS Driver

38.4.1 Overview

Data Structures

- struct [uart_rtos_config_t](#)
UART configuration structure. [More...](#)

Driver version

- #define [FSL_UART_FREERTOS_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 3, 0))
UART FreeRTOS driver version 2.3.0.

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, const 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.

38.4.2 Data Structure Documentation

38.4.2.1 struct [uart_rtos_config_t](#)

Data Fields

- UART_Type * [base](#)
UART base address.
- uint32_t [srcclk](#)
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.

38.4.3 Macro Definition Documentation

38.4.3.1 `#define FSL_UART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))`

38.4.4 Function Documentation

38.4.4.1 `int UART_RTOS_Init (uart_rtos_handle_t * handle, uart_handle_t * t_handle, const uart_rtos_config_t * cfg)`

Parameters

<i>handle</i>	The RTOS UART handle, the pointer to an allocated space for RTOS context.
<i>t_handle</i>	The pointer to the allocated space to store the transactional layer internal state.
<i>cfg</i>	The pointer to the parameters required to configure the UART after initialization.

Returns

0 succeed; otherwise fail.

38.4.4.2 `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

<i>handle</i>	The RTOS UART handle.
---------------	-----------------------

38.4.4.3 `int UART_RTOS_Send (uart_rtos_handle_t * handle, const uint8_t * buffer, uint32_t length)`

This function sends data. It is a synchronous API. If the hardware buffer is full, the task is in the blocked state.

UART FreeRTOS Driver

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.

38.4.4.4 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.

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.

38.5.1 UART CMSIS Driver

38.5.1.1 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)
{
}
```

38.5.1.2 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)
```

UART CMSIS Driver

```
    {
        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 39

VREF: Voltage Reference Driver

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.

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.

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.

Function Documentation

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.

Data Structure Documentation

39.4.1 struct vref_config_t

Data Fields

- [vref_buffer_mode_t](#) bufferMode
Buffer mode selection.

Macro Definition Documentation

39.5.1 #define FSL_VREF_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))

Enumeration Type Documentation

39.6.1 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.

Function Documentation

39.7.1 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.

39.7.2 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.
-------------	--------------------------

39.7.3 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.
---------------	--

39.7.4 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.

Function Documentation

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)).

39.7.5 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 40

WDOG: Watchdog Timer Driver

Overview

The MCUXpresso SDK provides a peripheral driver for the Watchdog module (WDOG) of MCUXpresso SDK devices.

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_ByteTest` = 1U }
Describes WDOG test mode.
- enum `wdog_tested_byte_t` {
 `kWDOG_TestByte0` = 0U,
 `kWDOG_TestByte1` = 1U,
 `kWDOG_TestByte2` = 2U,

Typical use case

`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.

Data Structure Documentation

40.3.1 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.

40.3.2 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.

Enumeration Type Documentation

- uint32_t [windowValue](#)
Window value.
- uint32_t [timeoutValue](#)
Timeout value.

40.3.3 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.

Macro Definition Documentation

40.4.1 #define FSL_WDOG_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

Enumeration Type Documentation

40.5.1 enum wdog_clock_source_t

Enumerator

kWDOG_LpoClockSource WDOG clock sourced from LPO.
kWDOG_AlternateClockSource WDOG clock sourced from alternate clock source.

40.5.2 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.

40.5.3 enum wdog_test_mode_t

Enumerator

kWDOG_QuickTest Selects quick test.

kWDOG_ByteTest Selects byte test.

40.5.4 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.

40.5.5 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.

40.5.6 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.

Function Documentation

40.6.1 void WDOG_GetDefaultConfig (wdog_config_t * config)

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

Function Documentation

```
* wdogConfig->enableWdog = true;
* wdogConfig->clockSource = kWDOG_LpoClockSource;
* 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](#)

40.6.2 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

40.6.3 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.

40.6.4 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

40.6.5 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
-------------	------------------------------

40.6.6 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

Function Documentation

<i>base</i>	WDOG peripheral base address
-------------	------------------------------

40.6.7 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

40.6.8 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

40.6.9 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.

40.6.10 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

40.6.11 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.

Function Documentation

Parameters

<i>base</i>	WDOG peripheral base address
<i>timeoutCount</i>	WDOG timeout value; count of WDOG clock tick.

40.6.12 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.

40.6.13 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
-------------	------------------------------

40.6.14 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
-------------	------------------------------

40.6.15 `static uint16_t WDOG_GetResetCount (WDOG_Type * base) [inline],
[static]`

This function gets the WDOG reset count value.

Function Documentation

Parameters

<i>base</i>	WDOG peripheral base address
-------------	------------------------------

Returns

WDOG reset count value.

40.6.16 `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 41

Debug Console

Overview

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

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

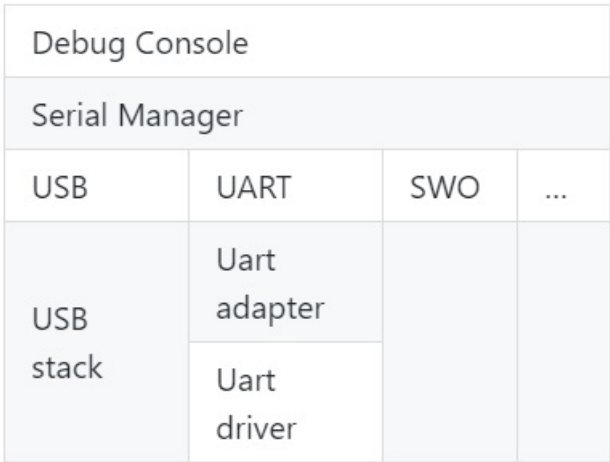


Figure 41.1.1: Debug console overview

Function groups

41.2.1 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,
```

Function groups

```
kSerialPort_UsbCdc,  
kSerialPort_Swo,  
kSerialPort_UsbCdcVirtual,  
} serial_port_type_t;
```

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

This example shows how to call the `DbgConsole_Init()` given the user configuration structure.

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

41.2.2 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

Function groups

specifier	Description
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).

length	Description
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 */
```

Typical use case

```
#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 */
```

NOTE: The macro `SDK_DEBUGCONSOLE_UART` 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_readc` will be used when `__REDLIB` is defined;** `_write` and `_read` will be used in other cases. If the macro `SDK_DEBUGCONSOLE_UART` is not defined, the semihosting will be used.

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 equivalent 255", 255);
```

Statement prints the decimal floating point and unsigned decimal.

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

Some examples use the SCANF function

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

Print out failure messages using MCUXpresso SDK `__assert_func`:

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

Note:

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

Modules

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

Macros

- #define [DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN](#) 0U
Definition select redirect toolchain printf, scanf to uart or not.
- #define [DEBUGCONSOLE_REDIRECT_TO_SDK](#) 1U
Select SDK version printf, scanf.
- #define [DEBUGCONSOLE_DISABLE](#) 2U
Disable debugconsole function.
- #define [SDK_DEBUGCONSOLE](#) 1U
Definition to select sdk or toolchain printf, scanf.
- #define [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

Macro Definition Documentation

Initialization

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

Macro Definition Documentation

41.4.1 #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U

Select toolchain printf and scanf.

41.4.2 #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U

41.4.3 #define DEBUGCONSOLE_DISABLE 2U

41.4.4 #define SDK_DEBUGCONSOLE 1U

The macro only support to be redefined in project setting.

41.4.5 #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.

Function Documentation

41.5.1 `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.
<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 • kSerialPort_UsbCdcVirtual.
<i>clkSrcFreq</i>	Frequency of peripheral source clock.

Returns

Indicates whether initialization was successful or not.

Return values

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

41.5.2 `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.

41.5.3 `int DbgConsole_Printf (const char * formatString, ...)`

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

Function Documentation

Parameters

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

Returns

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

41.5.4 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.

41.5.5 int DbgConsole_Scanf (char * *formatString*, ...)

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

Note

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

Parameters

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

Returns

Returns the number of fields successfully converted and assigned.

41.5.6 int DbgConsole_Getchar (void)

Call this function to read a character from standard input.

Note

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

Returns

Returns the character read.

41.5.7 int DbgConsole_BlockingPrintf (const char * *formatString*, ...)

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

Returns

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

41.5.8 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.

Function Documentation

41.5.9 int StrFormatPrintf (const char * *fmt*, va_list *ap*, char * *buf*, printfCb *cb*)

Note

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

Parameters

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

Returns

Number of characters to be print

41.5.10 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 "".
---------------	-----------------------------------

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.

41.6.1 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.

41.6.2 Guide Semihosting for Keil µVision

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

41.6.3 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.

41.6.4 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

Semihosting

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 = MK64FN1M0xxx12
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.

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.

41.7.1 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

41.7.1.1 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.

SWO

NOTE: Case a or c only apply at example which enable swo function,the SDK_DEBUGCONSOLE_UART definition in fsl_debug_console.h. For case a and c, Do and not do the above third step will be not affect function.

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

Step 2: Building the project

Step 3: Starting swo

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

41.7.2 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 O-K, please make sure the Core clock is set correctly, enable autodetect max SWO clk, enable ITM Stimulus Ports 0.

Step 3: Building the project

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

Step 4: Run the project

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

41.7.3 Guide SWO for MCUXpresso IDE

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

41.7.4 Guide SWO for ARMGCC

NOTE: ARMGCC has no library support SWO.

Chapter 42

Notification Framework

Overview

This section describes the programming interface of the Notifier driver.

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)
```

Notifier Overview

```
{
    ...
    ...
    ...
}
...
...
...
...
...
// 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.

Data Structure Documentation

42.3.1 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`

Data Structure Documentation

Configure notification type.

42.3.1.0.0.42 Field Documentation

42.3.1.0.0.42.1 `notifier_user_config_t* notifier_notification_block_t::targetConfig`

42.3.1.0.0.42.2 `notifier_policy_t notifier_notification_block_t::policy`

42.3.1.0.0.42.3 `notifier_notification_type_t notifier_notification_block_t::notifyType`

42.3.2 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.

42.3.2.0.0.43 Field Documentation

42.3.2.0.0.43.1 `notifier_callback_t notifier_callback_config_t::callback`

42.3.2.0.0.43.2 `notifier_callback_type_t notifier_callback_config_t::callbackType`

42.3.2.0.0.43.3 `void* notifier_callback_config_t::callbackData`

42.3.3 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.

42.3.3.0.0.44 Field Documentation

42.3.3.0.0.44.1 [notifier_user_config_t](#)** [notifier_handle_t::configsTable](#)

42.3.3.0.0.44.2 [uint8_t](#) [notifier_handle_t::configsNumber](#)

42.3.3.0.0.44.3 [notifier_callback_config_t](#)* [notifier_handle_t::callbacksTable](#)

42.3.3.0.0.44.4 [uint8_t](#) [notifier_handle_t::callbacksNumber](#)

42.3.3.0.0.44.5 [uint8_t](#) [notifier_handle_t::errorCallbackIndex](#)

42.3.3.0.0.44.6 [uint8_t](#) [notifier_handle_t::currentConfigIndex](#)

42.3.3.0.0.44.7 [notifier_user_function_t](#) [notifier_handle_t::userFunction](#)

42.3.3.0.0.44.8 [void](#)* [notifier_handle_t::userData](#)

Typedef Documentation

42.4.1 [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.

42.4.2 [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.

Enumeration Type Documentation

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`.

42.4.3 `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`.

Enumeration Type Documentation

42.5.1 `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.

42.5.2 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.

42.5.3 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.

42.5.4 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.

Function Documentation

42.6.1 `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.

42.6.2 **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

Function Documentation

<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.

42.6.3 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 43

Shell

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.

Function groups

43.2.1 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>");
```

43.2.2 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.

43.2.3 Shell Operation

```
SHELL_Init(s_shellHandle, s_serialHandle, "Test@SHELL>");  
SHELL_Task((s_shellHandle);
```

Function groups

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](#) (520U)
The handle size of the shell module.
- #define [SHELL_USE_COMMON_TASK](#) (1U)
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, paramCount)
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, 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.
- `void SHELL_Task (shell_handle_t shellHandle)`
The task function for Shell.

Data Structure Documentation

43.3.1 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

43.3.1.0.0.45 Field Documentation

43.3.1.0.0.45.1 const char* shell_command_t::pcCommand

For example "help". It must be all lower case.

43.3.1.0.0.45.2 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".

43.3.1.0.0.45.3 const cmd_function_t shell_command_t::pFuncCallBack

43.3.1.0.0.45.4 uint8_t shell_command_t::cExpectedNumberOfParameters

Macro Definition Documentation

43.4.1 #define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE

43.4.2 #define SHELL_AUTO_COMPLETE (1U)

43.4.3 #define SHELL_BUFFER_SIZE (64U)

43.4.4 #define SHELL_MAX_ARGS (8U)

43.4.5 #define SHELL_HISTORY_COUNT (3U)

43.4.6 #define SHELL_HANDLE_SIZE (520U)

It is the sum of the $SHELL_HISTORY_COUNT * SHELL_BUFFER_SIZE + SHELL_BUFFER_SIZE + SERIAL_MANAGER_READ_HANDLE_SIZE + SERIAL_MANAGER_WRITE_HANDLE_SIZE$

43.4.7 #define SHELL_USE_COMMON_TASK (1U)

43.4.8 #define SHELL_TASK_PRIORITY (2U)

43.4.9 #define SHELL_TASK_STACK_SIZE (1000U)

**43.4.10 #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.
-------------	--------------------------------------

43.4.11 #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.

43.4.12 #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.

Function Documentation

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".
----------------	--

Typedef Documentation

43.5.1 typedef shell_status_t(* cmd_function_t)(shell_handle_t shellHandle, int32_t argc, char **argv)

Enumeration Type Documentation

43.6.1 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.

Function Documentation

43.7.1 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_Open-WriteHandleFailed</i>	Open the write handle failed.
<i>kStatus_SHELL_Open-ReadHandleFailed</i>	Open the read handle failed.

43.7.2 **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.

43.7.3 **shell_status_t** SHELL_UnregisterCommand (**shell_command_t** * *shellCommand*)

This function is used to unregister the shell command.

Function Documentation

Parameters

<i>shellCommand</i>	The command element.
---------------------	----------------------

Return values

<i>kStatus_SHELL_Success</i>	Successfully unregister the command.
------------------------------	--------------------------------------

43.7.4 shell_status_t SHELL_Write (shell_handle_t *shellHandle*, 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.

43.7.5 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.

43.7.6 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.

Function Documentation

Parameters

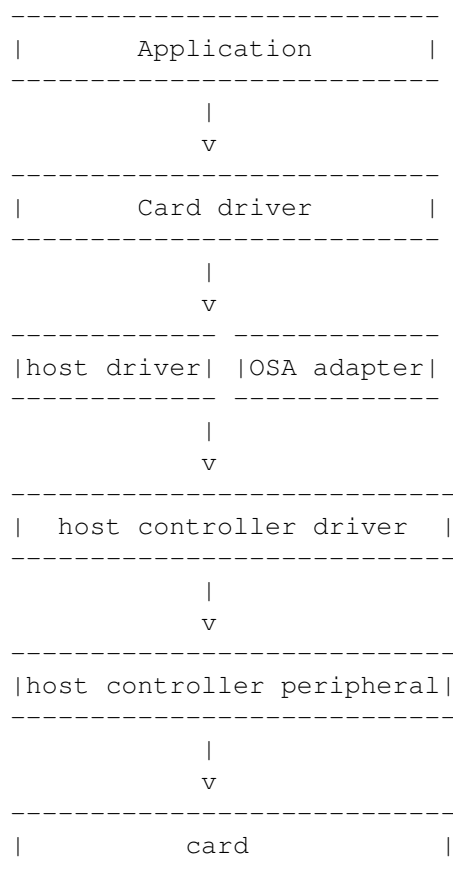
<i>shellHandle</i>	The shell module handle pointer.
--------------------	----------------------------------

Chapter 44

Secure Digital Card/Embedded MultiMedia Card/SDIO card

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 driver,



Modules

- [HOST Driver](#)
- [MMC Card Driver](#)
- [SD Card Driver](#)
- [SDIO Card Driver](#)
- [SDMMC Common](#)
- [SDMMC OSA](#)

SDIO Card Driver

SDIO Card Driver

44.2.1 Overview

The SDIO card driver provide card initialization/IO direct and extend command interface.

44.2.2 SDIO CARD Operation

error log support

Not support yet

User configurable

Board dependency

Typical use case

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

Data Structures

- struct [sdio_card_t](#)
SDIO card state. [More...](#)

Macros

- #define [FSL_SDIO_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2U, 3U, 0U)) /*2.3.0*/
Middleware version.
- #define [FSL_SDIO_MAX_IO_NUMS](#) (7U)
sdio device support maximum IO number

Typedefs

- typedef void(* [sdio_io_irq_handler_t](#))(sdio_card_t *card, uint32_t func)
sdio io handler

Enumerations

- enum [sdio_io_direction_t](#) {
 [kSDIO_IORead](#) = 0U,
 [kSDIO_IOWrite](#) = 1U }
}

sdio io read/write direction

Initialization and deinitialization

- [status_t SDIO_Init](#) (sdio_card_t *card)
SDIO card init function.
- void [SDIO_Deinit](#) (sdio_card_t *card)
SDIO card deinit, include card and host deinit.
- [status_t SDIO_CardInit](#) (sdio_card_t *card)
Initializes the card.
- void [SDIO_CardDeinit](#) (sdio_card_t *card)
Deinitializes the card.
- [status_t SDIO_HostInit](#) (sdio_card_t *card)
initialize the host.
- void [SDIO_HostDeinit](#) (sdio_card_t *card)
Deinitializes the host.
- void [SDIO_HostReset](#) (SDMMCHOST_CONFIG *host)
reset the host.
- void [SDIO_HostDoReset](#) (sdio_card_t *card)
reset the host.
- void [SDIO_PowerOnCard](#) (SDMMCHOST_TYPE *base, const [sdmmchost_pwr_card_t](#) *pwr)
power on card.
- void [SDIO_PowerOffCard](#) (SDMMCHOST_TYPE *base, const [sdmmchost_pwr_card_t](#) *pwr)
power on card.
- void [SDIO_SetCardPower](#) (sdio_card_t *card, bool enable)
set card power.
- [status_t SDIO_CardInActive](#) (sdio_card_t *card)
set SDIO card to inactive state
- [status_t SDIO_GetCardCapability](#) (sdio_card_t *card, [sdio_func_num_t](#) func)
get SDIO card capability
- [status_t SDIO_SetBlockSize](#) (sdio_card_t *card, [sdio_func_num_t](#) func, uint32_t blockSize)
set SDIO card block size
- [status_t SDIO_CardReset](#) (sdio_card_t *card)
set SDIO card reset
- [status_t SDIO_SetDataBusWidth](#) (sdio_card_t *card, [sdio_bus_width_t](#) busWidth)
set SDIO card data bus width
- [status_t SDIO_SwitchToHighSpeed](#) (sdio_card_t *card)
switch the card to high speed
- [status_t SDIO_ReadCIS](#) (sdio_card_t *card, [sdio_func_num_t](#) func, const uint32_t *tupleList, uint32_t tupleNum)
read SDIO card CIS for each function
- [status_t SDIO_WaitCardDetectStatus](#) (SDMMCHOST_TYPE *hostBase, const [sdmmchost_detect_card_t](#) *cd, bool waitCardStatus)
sdio wait card detect function.
- [status_t SDIO_PollingCardInsert](#) (sdio_card_t *card, uint32_t status)
sdio wait card detect function.
- bool [SDIO_IsCardPresent](#) (sdio_card_t *card)
sdio card present check function.

IO operations

- [status_t SDIO_IO_Write_Direct](#) (sdio_card_t *card, [sdio_func_num_t](#) func, uint32_t regAddr, uint8_t *data, bool raw)
IO direct write transfer function.
- [status_t SDIO_IO_Read_Direct](#) (sdio_card_t *card, [sdio_func_num_t](#) func, uint32_t regAddr, uint8_t *data)
IO direct read transfer function.
- [status_t SDIO_IO_RW_Direct](#) (sdio_card_t *card, [sdio_io_direction_t](#) direction, [sdio_func_num_t](#) func, uint32_t regAddr, uint8_t dataIn, uint8_t *dataOut)
IO direct read/write transfer function.
- [status_t SDIO_IO_Write_Extended](#) (sdio_card_t *card, [sdio_func_num_t](#) func, uint32_t regAddr, uint8_t *buffer, uint32_t count, uint32_t flags)
IO extended write transfer function.
- [status_t SDIO_IO_Read_Extended](#) (sdio_card_t *card, [sdio_func_num_t](#) func, uint32_t regAddr, uint8_t *buffer, uint32_t count, uint32_t flags)
IO extended read transfer function.
- [status_t SDIO_EnableIOInterrupt](#) (sdio_card_t *card, [sdio_func_num_t](#) func, bool enable)
enable IO interrupt
- [status_t SDIO_EnableIO](#) (sdio_card_t *card, [sdio_func_num_t](#) func, bool enable)
enable IO and wait IO ready
- [status_t SDIO_SelectIO](#) (sdio_card_t *card, [sdio_func_num_t](#) func)
select IO
- [status_t SDIO_AbortIO](#) (sdio_card_t *card, [sdio_func_num_t](#) func)
Abort IO transfer.
- [status_t SDIO_SetDriverStrength](#) (sdio_card_t *card, [sd_driver_strength_t](#) driverStrength)
Set driver strength.
- [status_t SDIO_EnableAsyncInterrupt](#) (sdio_card_t *card, bool enable)
Enable/Disable Async interrupt.
- [status_t SDIO_GetPendingInterrupt](#) (sdio_card_t *card, uint8_t *pendingInt)
Get pending interrupt.
- [status_t SDIO_IO_Transfer](#) (sdio_card_t *card, [sdio_command_t](#) cmd, uint32_t argument, uint32_t blockSize, uint8_t *txData, uint8_t *rxData, uint16_t dataSize, uint32_t *response)
sdio card io transfer function.
- void [SDIO_SetIOIRQHandler](#) (sdio_card_t *card, [sdio_func_num_t](#) func, [sdio_io_irq_handler_t](#) handler)
sdio set io IRQ handler.
- [status_t SDIO_HandlePendingIOInterrupt](#) (sdio_card_t *card)
sdio card io pending interrupt handle function.

44.2.3 Data Structure Documentation

44.2.3.1 struct _sdio_card

sdio card descriptor

Define the card structure including the necessary fields to identify and describe the card.

Data Fields

- `sdrmchost_t * host`
Host information.
- `sdio_usr_param_t usrParam`
user parameter
- `bool noInternalAlign`
use this flag to disable sdmmc align.
- `uint8_t internalBuffer [FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE]`
internal buffer
- `bool isHostReady`
use this flag to indicate if need host re-init or not
- `bool memPresentFlag`
indicate if memory present
- `uint32_t busClock_Hz`
SD bus clock frequency united in Hz.
- `uint32_t relativeAddress`
Relative address of the card.
- `uint8_t sdVersion`
SD version.
- `sd_timing_mode_t currentTiming`
current timing mode
- `sd_driver_strength_t driverStrength`
driver strength
- `sd_max_current_t maxCurrent`
card current limit
- `sdmmc_operation_voltage_t operationVoltage`
card operation voltage
- `uint8_t sdioVersion`
SDIO version.
- `uint8_t cccrVersion`
CCCR version.
- `uint8_t ioTotalNumber`
total number of IO function
- `uint32_t cccrflags`
Flags in _sd_card_flag.
- `uint32_t io0blockSize`
record the io0 block size
- `uint32_t ocr`
Raw OCR content, only 24bit available for SDIO card.
- `uint32_t commonCISPointer`
point to common CIS
- `sdio_common_cis_t commonCIS`
CIS table.
- `sdio_fbr_t ioFBR [FSL_SDIO_MAX_IO_NUMS]`
FBR table.
- `sdio_func_cis_t funcCIS [FSL_SDIO_MAX_IO_NUMS]`
function CIS table
- `sdio_io_irq_handler_t ioIRQHandler [FSL_SDIO_MAX_IO_NUMS]`
io IRQ handler
- `uint8_t ioIntIndex`

SDIO Card Driver

- used to record current enabled io interrupt index*
 - uint8_t [ioIntNums](#)*used to record total enabled io interrupt numbers*

44.2.3.1.0.46 Field Documentation

44.2.3.1.0.46.1 bool sdio_card_t::noInternalAlign

If disable, sdmmc will not make sure the data buffer address is word align, otherwise all the transfer are align to low level driver

44.2.4 Macro Definition Documentation

44.2.4.1 #define FSL_SDIO_DRIVER_VERSION (MAKE_VERSION(2U, 3U, 0U)) /*2.3.0*/

44.2.5 Enumeration Type Documentation

44.2.5.1 enum sdio_io_direction_t

Enumerator

kSDIO_IORead io read
kSDIO_IOWrite io write

44.2.6 Function Documentation

44.2.6.1 status_t SDIO_Init (sdio_card_t * card)

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_Go-IdleFailed</i>	
<i>kStatus_SDMMC_Hand-ShakeOperation-ConditionFailed</i>	

<i>kStatus_SDMMC_SDIO- _InvalidCard</i>	
<i>kStatus_SDMMC_SDIO- _InvalidVoltage</i>	
<i>kStatus_SDMMC_Send- RelativeAddressFailed</i>	
<i>kStatus_SDMMC_Select- CardFailed</i>	
<i>kStatus_SDMMC_SDIO- _SwitchHighSpeedFail</i>	
<i>kStatus_SDMMC_SDIO- _ReadCISFail</i>	
<i>kStatus_SDMMC_- TransferFailed</i>	
<i>kStatus_Success</i>	

44.2.6.2 void SDIO_Deinit (sdio_card_t * card)

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

44.2.6.3 status_t SDIO_CardInit (sdio_card_t * card)

This function initializes the card only, make sure the host is ready when call this function, otherwise it will return kStatus_SDMMC_HostNotReady.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_Host- NotReady</i>	host is not ready.
---	--------------------

SDIO Card Driver

<i>kStatus_SDMMC_GoIdleFailed</i>	Go idle failed.
<i>kStatus_SDMMC_NotSupportYet</i>	Card not support.
<i>kStatus_SDMMC_SendOperationConditionFailed</i>	Send operation condition failed.
<i>kStatus_SDMMC_AllSendCidFailed</i>	Send CID failed.
<i>kStatus_SDMMC_SendRelativeAddressFailed</i>	Send relative address failed.
<i>kStatus_SDMMC_SendCsdFailed</i>	Send CSD failed.
<i>kStatus_SDMMC_SelectCardFailed</i>	Send SELECT_CARD command failed.
<i>kStatus_SDMMC_SendScrFailed</i>	Send SCR failed.
<i>kStatus_SDMMC_SetBusWidthFailed</i>	Set bus width failed.
<i>kStatus_SDMMC_SwitchHighSpeedFailed</i>	Switch high speed failed.
<i>kStatus_SDMMC_SetCardBlockSizeFailed</i>	Set card block size failed.
<i>kStatus_Success</i>	Operate successfully.

44.2.6.4 void SDIO_CardDeinit (sdio_card_t * card)

This function deinitializes the specific card.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

44.2.6.5 status_t SDIO_HostInit (sdio_card_t * card)

This function deinitializes the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

44.2.6.6 void SDIO_HostDeinit (sdio_card_t * *card*)

This function deinitializes the host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

44.2.6.7 void SDIO_HostReset (SDMMCHOST_CONFIG * *host*)

This function reset the specific host.

Deprecated Do not use this function. It has been superceded by [SDIO_HostDoReset](#).

Parameters

<i>host</i>	host descriptor.
-------------	------------------

44.2.6.8 void SDIO_HostDoReset (sdio_card_t * *card*)

This function reset the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

44.2.6.9 void SDIO_PowerOnCard (SDMMCHOST_TYPE * *base*, const sdmmchost_pwr_card_t * *pwr*)

The power on operation depend on host or the user define power on function.

Deprecated Do not use this function. It has been superceded by [SDIO_SetCardPower](#).

SDIO Card Driver

Parameters

<i>base</i>	host base address.
<i>pwr</i>	user define power control configuration

44.2.6.10 void SDIO_PowerOffCard (SDMMCHOST_TYPE * *base*, const sdmmchost_pwr_card_t * *pwr*)

The power off operation depend on host or the user define power on function.

Deprecated Do not use this function. It has been superceded by [SDIO_SetCardPower](#).

Parameters

<i>base</i>	host base address.
<i>pwr</i>	user define power control configuration

44.2.6.11 void SDIO_SetCardPower (sdio_card_t * *card*, bool *enable*)

The power off operation depend on host or the user define power on function.

Parameters

<i>card</i>	card descriptor.
<i>enable</i>	true is power on, false is power off.

44.2.6.12 status_t SDIO_CardIsActive (sdio_card_t * *card*)

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
-------------------------------------	--

<i>kStatus_Success</i>	
------------------------	--

44.2.6.13 status_t SDIO_GetCardCapability (sdio_card_t * *card*, sdio_func_num_t *func*)

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_Success</i>	

44.2.6.14 status_t SDIO_SetBlockSize (sdio_card_t * *card*, sdio_func_num_t *func*, uint32_t *blockSize*)

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	io number
<i>blockSize</i>	block size

Return values

<i>kStatus_SDMMC_Set-CardBlockSizeFailed</i>	
<i>kStatus_SDMMC_SDIO-_InvalidArgument</i>	
<i>kStatus_Success</i>	

44.2.6.15 status_t SDIO_CardReset (sdio_card_t * *card*)

SDIO Card Driver

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_Success</i>	

44.2.6.16 **status_t SDIO_SetDataBusWidth (sdio_card_t * *card*, sdio_bus_width_t *busWidth*)**

Parameters

<i>card</i>	Card descriptor.
<i>busWidth</i>	bus width

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_Success</i>	

44.2.6.17 **status_t SDIO_SwitchToHighSpeed (sdio_card_t * *card*)**

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
--------------------------------------	--

<i>kStatus_SDMMC_SDIO- _SwitchHighSpeedFail</i>	
<i>kStatus_Success</i>	

44.2.6.18 status_t SDIO_ReadCIS (sdio_card_t * *card*, sdio_func_num_t *func*, const uint32_t * *tupleList*, uint32_t *tupleNum*)

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	io number
<i>tupleList</i>	code list
<i>tupleNum</i>	code number

Return values

<i>kStatus_SDMMC_SDIO- _ReadCISFail</i>	
<i>kStatus_SDMMC_- TransferFailed</i>	
<i>kStatus_Success</i>	

44.2.6.19 status_t SDIO_WaitCardDetectStatus (SDMMCHOST_TYPE * *hostBase*, const sdmmchost_detect_card_t * *cd*, bool *waitCardStatus*)

Detect card through GPIO, CD, DATA3.

Deprecated Do not use this function. It has been superseded by [SDIO_PollingCardInsert](#).

Parameters

<i>hostBase</i>	card descriptor.
<i>cd</i>	detect configuration

SDIO Card Driver

<i>waitCardStatus</i>	wait card detect status
-----------------------	-------------------------

44.2.6.20 **status_t SDIO_PollingCardInsert (sdio_card_t * *card*, uint32_t *status*)**

Detect card through GPIO, CD, DATA3.

Parameters

<i>card</i>	card descriptor.
<i>status</i>	detect status, kSD_Inserted or kSD_Removed.

44.2.6.21 **bool SDIO_IsCardPresent (sdio_card_t * *card*)**

Parameters

<i>card</i>	card descriptor.
-------------	------------------

44.2.6.22 **status_t SDIO_IO_Write_Direct (sdio_card_t * *card*, sdio_func_num_t *func*, uint32_t *regAddr*, uint8_t * *data*, bool *raw*)**

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO numner
<i>regAddr</i>	register address
<i>data</i>	the data pinter to write
<i>raw</i>	flag, indicate read after write or write only

Return values

<i>kStatus_SDMMC_- TransferFailed</i>	
---	--

<i>kStatus_Success</i>	
------------------------	--

44.2.6.23 `status_t SDIO_IO_Read_Direct (sdio_card_t * card, sdio_func_num_t func,
uint32_t regAddr, uint8_t * data)`

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number
<i>regAddr</i>	register address
<i>data</i>	pointer to read

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_Success</i>	

44.2.6.24 `status_t SDIO_IO_RW_Direct (sdio_card_t * card, sdio_io_direction_t direction,
sdio_func_num_t func, uint32_t regAddr, uint8_t dataIn, uint8_t * dataOut)`

Parameters

<i>card</i>	Card descriptor.
<i>direction</i>	io access direction, please reference sdio_io_direction_t.
<i>func</i>	IO number
<i>regAddr</i>	register address
<i>dataIn</i>	data to write
<i>dataOut</i>	data pointer for readback data, support both for read and write, when application want readback the data after write command, dataOut should not be NULL.

Return values

SDIO Card Driver

<i>kStatus_SDMMC_- TransferFailed</i>	
<i>kStatus_Success</i>	

44.2.6.25 `status_t SDIO_IO_Write_Extended (sdio_card_t * card, sdio_func_num_t func,
uint32_t regAddr, uint8_t * buffer, uint32_t count, uint32_t flags)`

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number
<i>regAddr</i>	register address
<i>buffer</i>	data buffer to write
<i>count</i>	data count
<i>flags</i>	write flags

Return values

<i>kStatus_SDMMC_- TransferFailed</i>	
<i>kStatus_SDMMC_SDIO- _InvalidArgument</i>	
<i>kStatus_Success</i>	

44.2.6.26 `status_t SDIO_IO_Read_Extended (sdio_card_t * card, sdio_func_num_t func,
uint32_t regAddr, uint8_t * buffer, uint32_t count, uint32_t flags)`

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number
<i>regAddr</i>	register address
<i>buffer</i>	data buffer to read
<i>count</i>	data count
<i>flags</i>	write flags

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_SDMMC_SDIO-_InvalidArgument</i>	
<i>kStatus_Success</i>	

44.2.6.27 `status_t SDIO_EnableIOInterrupt (sdio_card_t * card, sdio_func_num_t func, bool enable)`

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number
<i>enable</i>	enable/disable flag

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_Success</i>	

44.2.6.28 `status_t SDIO_EnableIO (sdio_card_t * card, sdio_func_num_t func, bool enable)`

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number
<i>enable</i>	enable/disable flag

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_Success</i>	

44.2.6.29 `status_t SDIO_SelectIO (sdio_card_t * card, sdio_func_num_t func)`

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_Success</i>	

44.2.6.30 status_t SDIO_AbortIO (sdio_card_t * *card*, sdio_func_num_t *func*)

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_Success</i>	

44.2.6.31 status_t SDIO_SetDriverStrength (sdio_card_t * *card*, sd_driver_strength_t *driverStrength*)

Parameters

<i>card</i>	Card descriptor.
<i>driverStrength</i>	target driver strength.

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_Success</i>	

44.2.6.32 `status_t SDIO_EnableAsyncInterrupt (sdio_card_t * card, bool enable)`

Parameters

<i>card</i>	Card descriptor.
<i>enable</i>	true is enable, false is disable.

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_Success</i>	

44.2.6.33 status_t SDIO_GetPendingInterrupt (sdio_card_t * *card*, uint8_t * *pendingInt*)

Parameters

<i>card</i>	Card descriptor.
<i>pendingInt</i>	pointer store pending interrupt

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_Success</i>	

44.2.6.34 status_t SDIO_IO_Transfer (sdio_card_t * *card*, sdio_command_t *cmd*, uint32_t *argument*, uint32_t *blockSize*, uint8_t * *txData*, uint8_t * *rxData*, uint16_t *dataSize*, uint32_t * *response*)

This function can be used for transfer direct/extend command. Please pay attention to the non-align data buffer address transfer, if data buffer address can not meet host controller internal DMA requirement, sdio driver will try to use internal align buffer if data size is not bigger than internal buffer size, Align address transfer always can get a better performance, so if application want sdio driver make sure buffer address align, please redefine the SDMMC_GLOBAL_BUFFER_SIZE macro to a value which is big enough for your application.

SDIO Card Driver

Parameters

<i>card</i>	card descriptor.
<i>cmd</i>	command to transfer
<i>argument</i>	argument to transfer
<i>blockSize</i>	used for block mode.
<i>txData</i>	tx buffer pointer or NULL
<i>rxData</i>	rx buffer pointer or NULL
<i>dataSize</i>	transfer data size
<i>response</i>	reponse pointer, if application want read response back, please set it to a NON-NULL pointer.

44.2.6.35 void SDIO_SetIOIRQHandler (sdio_card_t * *card*, sdio_func_num_t *func*, sdio_io_irq_handler_t *handler*)

Parameters

<i>card</i>	card descriptor.
<i>func</i>	function io number.
<i>handler</i>	io IRQ handler.

44.2.6.36 status_t SDIO_HandlePendingIOInterrupt (sdio_card_t * *card*)

This function is used to handle the pending io interrupt. To reigster a IO IRQ handler,

```
* //initialization
* SDIO_EnableIOInterrupt(card, 0, true);
* SDIO_SetIOIRQHandler(card, 0, func0_handler);
* //call it in interrupt callback
* SDIO_HandlePendingIOInterrupt(card);
*
```

To releae a IO IRQ handler,

```
* SDIO_EnableIOInterrupt(card, 0, false);
* SDIO_SetIOIRQHandler(card, 0, NULL);
*
```

Parameters

<i>card</i>	card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_</i> <i>TransferFailed</i>	
<i>kStatus_Success</i>	

SD Card Driver

SD Card Driver

44.3.1 Overview

The SDCARD driver provide card initialization/read/write/erase interface.

44.3.2 SD CARD Operation

error log support

Lots of error log has been added to sd relate functions, if error occurs during initial/read/write, please enable the error log print functionality with `#define SDMMC_ENABLE_LOG_PRINT 1` And rerun the project then user can check what kind of error happened.

User configurable

```
typedef struct _sd_card
{
    sdmmchost_t *host;
    sd_usr_param_t usrParam;
    bool isHostReady;
    bool noInternalAlign;
    uint32_t busClock_Hz;
    uint32_t relativeAddress;
    uint32_t version;
    uint32_t flags;
    uint8_t internalBuffer[FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE];
    uint32_t ocr;
    sd_cid_t cid;
    sd_csd_t csd;
    sd_scr_t scr;
    sd_status_t stat;
    uint32_t blockCount;
    uint32_t blockSize;
    sd_timing_mode_t currentTiming;
    sd_driver_strength_t driverStrength;
    sd_max_current_t maxCurrent;
    sdmmc_operation_voltage_t operationVoltage;
} sd_card_t;
```

Part of The variables above is user configurable,

1. host Application need to provide host controller base address and the host's source clock frequency, etc. For example:

```
/* allocate dma descriptor buffer for host controller */
s_host.dmaDesBuffer = s_sdmmcHostDmaBuffer;
s_host.dmaDesBufferWordsNum = xxx;
/* */
((sd_card_t *)card)->host = &s_host;
((sd_card_t *)card)->host->hostController.base = BOARD_SDMMC_SD_HOST_BASEADDR;
((sd_card_t *)card)->host->hostController.sourceClock_Hz = BOARD_USDHC1ClockConfiguration();

/* allocate resource for sdmmc osa layer */
((sd_card_t *)card)->host->hostEvent = &s_event;
```


2. sdcard_usr_param_t usrParam

```
/* board layer configuration register */
((sd_card_t *)card)->usrParam.cd          = &s_cd;
((sd_card_t *)card)->usrParam.pwr         = BOARD_SDCardPowerControl;
((sd_card_t *)card)->usrParam.ioStrength = BOARD_SD_Pin_Config;
((sd_card_t *)card)->usrParam.ioVoltage  = &s_ioVoltage;
((sd_card_t *)card)->usrParam.maxFreq    = BOARD_SDMMC_SD_HOST_SUPPORT_SDR104_FREQ;
```

a. cd-which allow application define the card insert/remove callback function, redefine the card detect timeout ms and also allow application determine how to detect card.

b. pwr-which allow application redefine the card power on/off function.

c. ioStrength-which is used to switch the signal pin configurations include driver strength/speed mode dynamically for different timing(SDR/HS timing) mode, reference the function defined sdmmc_config.c

d. ioVoltage-which allow application register io voltage switch function instead of using the function host driver provided for SDR/HS200/HS400 timing.

e. maxFreq-which allow application set the maximum bus clock that the board support.

3. bool noInternalAlign

Sdmmc include an address align internal buffer(to use host controller internal DMA), to improve

4. sd_timing_mode_t currentTiming

It is used to indicate the currentTiming the card is working on, however sdmmc also support p Generally, user may not set this variable if you don't know what kind of timing the card support

5. sd_driver_strength_t driverStrength

Choose a valid card driver strength if application required and call SD_SetDriverStrength in

6. sd_max_current_t maxCurrent

Choose a valid card current if application required and call SD_SetMaxCurrent in application

Typical use case

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

Data Structures

- struct `sd_card_t`
SD card state. [More...](#)

Macros

- #define `FSL_SD_DRIVER_VERSION` (MAKE_VERSION(2U, 3U, 0U)) /*2.3.0*/
Driver version.

Enumerations

- enum `_sd_card_flag` {
 `kSD_SupportHighCapacityFlag` = (1U << 1U),
 `kSD_Support4BitWidthFlag` = (1U << 2U),
 `kSD_SupportSdhcFlag` = (1U << 3U),
 `kSD_SupportSdxcFlag` = (1U << 4U),
 `kSD_SupportVoltage180v` = (1U << 5U),
 `kSD_SupportSetBlockCountCmd` = (1U << 6U),
 `kSD_SupportSpeedClassControlCmd` = (1U << 7U) }

SD card flags.

SDCARD Function

- `status_t SD_Init (sd_card_t *card)`
Initializes the card on a specific host controller.
- `void SD_Deinit (sd_card_t *card)`
Deinitializes the card.
- `status_t SD_CardInit (sd_card_t *card)`
Initializes the card.
- `void SD_CardDeinit (sd_card_t *card)`
Deinitializes the card.
- `status_t SD_HostInit (sd_card_t *card)`
initialize the host.
- `void SD_HostDeinit (sd_card_t *card)`
Deinitializes the host.
- `void SD_HostDoReset (sd_card_t *card)`
reset the host.
- `void SD_HostReset (SDMMCHOST_CONFIG *host)`
reset the host.
- `void SD_PowerOnCard (SDMMCHOST_TYPE *base, const sdmmchost_pwr_card_t *pwr)`
power on card.
- `void SD_PowerOffCard (SDMMCHOST_TYPE *base, const sdmmchost_pwr_card_t *pwr)`
power off card.
- `void SD_SetCardPower (sd_card_t *card, bool enable)`
set card power.
- `status_t SD_WaitCardDetectStatus (SDMMCHOST_TYPE *hostBase, const sdmmchost_detect_card_t *cd, bool waitCardStatus)`
sd wait card detect function.
- `status_t SD_PollingCardInsert (sd_card_t *card, uint32_t status)`
sd wait card detect function.
- `bool SD_IsCardPresent (sd_card_t *card)`
sd card present check function.
- `bool SD_CheckReadOnly (sd_card_t *card)`
Checks whether the card is write-protected.
- `status_t SD_SelectCard (sd_card_t *card, bool isSelected)`
Send SELECT_CARD command to set the card to be transfer state or not.
- `status_t SD_ReadStatus (sd_card_t *card)`
Send ACMD13 to get the card current status.

- `status_t SD_ReadBlocks (sd_card_t *card, uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)`
Reads blocks from the specific card.
- `status_t SD_WriteBlocks (sd_card_t *card, const uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)`
Writes blocks of data to the specific card.
- `status_t SD_EraseBlocks (sd_card_t *card, uint32_t startBlock, uint32_t blockCount)`
Erases blocks of the specific card.
- `status_t SD_SetDriverStrength (sd_card_t *card, sd_driver_strength_t driverStrength)`
select card driver strength select card driver strength
- `status_t SD_SetMaxCurrent (sd_card_t *card, sd_max_current_t maxCurrent)`
select max current select max operation current

44.3.3 Data Structure Documentation

44.3.3.1 struct sd_card_t

Define the card structure including the necessary fields to identify and describe the card.

Data Fields

- `sdmhost_t * host`
Host configuration.
- `sd_usr_param_t usrParam`
user parameter
- `bool isHostReady`
use this flag to indicate if need host re-init or not
- `bool noInternalAlign`
used to enable/disable the functionality of the exchange buffer
- `uint32_t busClock_Hz`
SD bus clock frequency united in Hz.
- `uint32_t relativeAddress`
Relative address of the card.
- `uint32_t version`
Card version.
- `uint32_t flags`
Flags in _sd_card_flag.
- `uint8_t internalBuffer [FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE]`
internal buffer
- `uint32_t ocr`
Raw OCR content.
- `sd_cid_t cid`
CID.
- `sd_csd_t csd`
CSD.
- `sd_scr_t scr`
SCR.
- `sd_status_t stat`

SD Card Driver

- sd 512 bit status*
- uint32_t [blockCount](#)
Card total block number.
- uint32_t [blockSize](#)
Card block size.
- [sd_timing_mode_t](#) [currentTiming](#)
current timing mode
- [sd_driver_strength_t](#) [driverStrength](#)
driver strength
- [sd_max_current_t](#) [maxCurrent](#)
card current limit
- [sdmmc_operation_voltage_t](#) [operationVoltage](#)
card operation voltage

44.3.4 Macro Definition Documentation

44.3.4.1 **#define FSL_SD_DRIVER_VERSION (MAKE_VERSION(2U, 3U, 0U)) /*2.3.0*/**

44.3.5 Enumeration Type Documentation

44.3.5.1 enum _sd_card_flag

Enumerator

kSD_SupportHighCapacityFlag Support high capacity.
kSD_Support4BitWidthFlag Support 4-bit data width.
kSD_SupportSdhcFlag Card is SDHC.
kSD_SupportSdxcFlag Card is SDXC.
kSD_SupportVoltage180v card support 1.8v voltage
kSD_SupportSetBlockCountCmd card support cmd23 flag
kSD_SupportSpeedClassControlCmd card support speed class control flag

44.3.6 Function Documentation

44.3.6.1 status_t SD_Init (sd_card_t * *card*)

Deprecated Do not use this function. It has been superceded by [SD_HostInit](#), [SD_CardInit](#).

This function initializes the card on a specific host controller, it is consist of host init, card detect, card init function, however user can ignore this high level function, instead of use the low level function, such as [SD_CardInit](#), [SD_HostInit](#), [SD_CardDetect](#).

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_Host-NotReady</i>	host is not ready.
<i>kStatus_SDMMC_Go-IdleFailed</i>	Go idle failed.
<i>kStatus_SDMMC_Not-SupportYet</i>	Card not support.
<i>kStatus_SDMMC_Send-OperationCondition-Failed</i>	Send operation condition failed.
<i>kStatus_SDMMC_All-SendCidFailed</i>	Send CID failed.
<i>kStatus_SDMMC_Send-RelativeAddressFailed</i>	Send relative address failed.
<i>kStatus_SDMMC_Send-CsdFailed</i>	Send CSD failed.
<i>kStatus_SDMMC_Select-CardFailed</i>	Send SELECT_CARD command failed.
<i>kStatus_SDMMC_Send-ScrFailed</i>	Send SCR failed.
<i>kStatus_SDMMC_SetBus-WidthFailed</i>	Set bus width failed.
<i>kStatus_SDMMC_Switch-HighSpeedFailed</i>	Switch high speed failed.
<i>kStatus_SDMMC_Set-CardBlockSizeFailed</i>	Set card block size failed.
<i>kStatus_Success</i>	Operate successfully.

44.3.6.2 void SD_Deinit (sd_card_t * card)

Deprecated Do not use this function. It has been superceded by [SD_HostDeinit](#), [SD_CardDeinit](#). This function deinitializes the specific card and host.

SD Card Driver

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

44.3.6.3 status_t SD_CardInit (sd_card_t * *card*)

This function initializes the card only, make sure the host is ready when call this function, otherwise it will return kStatus_SDMMC_HostNotReady.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_HostNotReady</i>	host is not ready.
<i>kStatus_SDMMC_GoIdleFailed</i>	Go idle failed.
<i>kStatus_SDMMC_NotSupportYet</i>	Card not support.
<i>kStatus_SDMMC_SendOperationConditionFailed</i>	Send operation condition failed.
<i>kStatus_SDMMC_AllSendCidFailed</i>	Send CID failed.
<i>kStatus_SDMMC_SendRelativeAddressFailed</i>	Send relative address failed.
<i>kStatus_SDMMC_SendCsdFailed</i>	Send CSD failed.
<i>kStatus_SDMMC_SelectCardFailed</i>	Send SELECT_CARD command failed.
<i>kStatus_SDMMC_SendScrFailed</i>	Send SCR failed.

<i>kStatus_SDMMC_SetBus-WidthFailed</i>	Set bus width failed.
<i>kStatus_SDMMC_Switch-HighSpeedFailed</i>	Switch high speed failed.
<i>kStatus_SDMMC_Set-CardBlockSizeFailed</i>	Set card block size failed.
<i>kStatus_Success</i>	Operate successfully.

44.3.6.4 void SD_CardDeinit (sd_card_t * *card*)

This function deinitializes the specific card.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

44.3.6.5 status_t SD_HostInit (sd_card_t * *card*)

This function deinitializes the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

44.3.6.6 void SD_HostDeinit (sd_card_t * *card*)

This function deinitializes the host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

44.3.6.7 void SD_HostDoReset (sd_card_t * *card*)

This function reset the specific host.

SD Card Driver

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

44.3.6.8 void SD_HostReset (SDMMCHOST_CONFIG * *host*)

This function reset the specific host.

Deprecated Do not use this function. It has been superceded by [SD_HostDoReset](#).

Parameters

<i>host</i>	host descriptor.
-------------	------------------

44.3.6.9 void SD_PowerOnCard (SDMMCHOST_TYPE * *base*, const sdmmchost_pwr_card_t * *pwr*)

The power on operation depend on host or the user define power on function.

Deprecated Do not use this function. It has been superceded by [SD_SetCardPower](#).

Parameters

<i>base</i>	host base address.
<i>pwr</i>	user define power control configuration

44.3.6.10 void SD_PowerOffCard (SDMMCHOST_TYPE * *base*, const sdmmchost_pwr_card_t * *pwr*)

The power off operation depend on host or the user define power on function.

Deprecated Do not use this function. It has been superceded by [SD_SetCardPower](#).

Parameters

<i>base</i>	host base address.
<i>pwr</i>	user define power control configuration

44.3.6.11 void SD_SetCardPower (sd_card_t * *card*, bool *enable*)

The power off operation depend on host or the user define power on function.

Parameters

<i>card</i>	card descriptor.
<i>enable</i>	true is power on, false is power off.

44.3.6.12 status_t SD_WaitCardDetectStatus (SDMMCHOST_TYPE * *hostBase*, const sdmmchost_detect_card_t * *cd*, bool *waitCardStatus*)

Detect card through GPIO, CD, DATA3.

Deprecated Do not use this function. It has been superceded by [SD_PollingCardInsert](#).

Parameters

<i>hostBase</i>	host base address.
<i>cd</i>	card detect configuration
<i>waitCardStatus</i>	wait card detect status

44.3.6.13 status_t SD_PollingCardInsert (sd_card_t * *card*, uint32_t *status*)

Detect card through GPIO, CD, DATA3.

Parameters

<i>card</i>	card descriptor.
<i>status</i>	detect status, kSD_Inserted or kSD_Removed.

44.3.6.14 bool SD_IsCardPresent (sd_card_t * *card*)

SD Card Driver

Parameters

<i>card</i>	card descriptor.
-------------	------------------

44.3.6.15 bool SD_CheckReadOnly (sd_card_t * *card*)

This function checks if the card is write-protected via the CSD register.

Parameters

<i>card</i>	The specific card.
-------------	--------------------

Return values

<i>true</i>	Card is read only.
<i>false</i>	Card isn't read only.

44.3.6.16 status_t SD_SelectCard (sd_card_t * *card*, bool *isSelected*)

Parameters

<i>card</i>	Card descriptor.
<i>isSelected</i>	True to set the card into transfer state, false to disselect.

Return values

<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_Success</i>	Operate successfully.

44.3.6.17 status_t SD_ReadStatus (sd_card_t * *card*)

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_SendApplicationCommandFailed</i>	send application command failed.
<i>kStatus_Success</i>	Operate successfully.

44.3.6.18 status_t SD_ReadBlocks (sd_card_t * *card*, uint8_t * *buffer*, uint32_t *startBlock*, uint32_t *blockCount*)

This function reads blocks from the specific card with default block size defined by the SDHC_CARD_DEFAULT_BLOCK_SIZE.

Parameters

<i>card</i>	Card descriptor.
<i>buffer</i>	The buffer to save 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_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_CardNotSupport</i>	Card not support.
<i>kStatus_SDMMC_NotSupportYet</i>	Not support now.
<i>kStatus_SDMMC_WaitWriteCompleteFailed</i>	Send status failed.
<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.

SD Card Driver

<i>kStatus_SDMMC_Stop-TransmissionFailed</i>	Stop transmission failed.
<i>kStatus_Success</i>	Operate successfully.

44.3.6.19 **status_t SD_WriteBlocks (sd_card_t * *card*, const uint8_t * *buffer*, uint32_t *startBlock*, uint32_t *blockCount*)**

This function writes blocks to the specific card with default block size 512 bytes.

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_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_Not-SupportYet</i>	Not support now.
<i>kStatus_SDMMC_Card-NotSupport</i>	Card not support.
<i>kStatus_SDMMC_Wait-WriteCompleteFailed</i>	Send status failed.
<i>kStatus_SDMMC_-TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_Stop-TransmissionFailed</i>	Stop transmission failed.
<i>kStatus_Success</i>	Operate successfully.

44.3.6.20 **status_t SD_EraseBlocks (sd_card_t * *card*, uint32_t *startBlock*, uint32_t *blockCount*)**

This function erases blocks of the specific card with default block size 512 bytes.

Parameters

<i>card</i>	Card descriptor.
<i>startBlock</i>	The start block index.
<i>blockCount</i>	The number of blocks to erase.

Return values

<i>kStatus_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_Wait-WriteCompleteFailed</i>	Send status failed.
<i>kStatus_SDMMC_-TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_Wait-WriteCompleteFailed</i>	Send status failed.
<i>kStatus_Success</i>	Operate successfully.

44.3.6.21 status_t SD_SetDriverStrength (sd_card_t * *card*, sd_driver_strength_t *driverStrength*)

Parameters

<i>card</i>	Card descriptor.
<i>driverStrength</i>	Driver strength

44.3.6.22 status_t SD_SetMaxCurrent (sd_card_t * *card*, sd_max_current_t *maxCurrent*)

Parameters

<i>card</i>	Card descriptor.
<i>maxCurrent</i>	Max current

MMC Card Driver

MMC Card Driver

44.4.1 Overview

The MMCCARD driver provide card initialization/read/write/erase interface.

44.4.2 MMC CARD Operation

error log support

Not support yet

User configurable

Board dependency

Typical use case

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

Data Structures

- struct [mmc_usr_param_t](#)
sdcard user parameter [More...](#)
- struct [mmc_card_t](#)
mmc card state [More...](#)

Macros

- #define [FSL_MMC_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2U, 3U, 0U)) /*2.3.0*/
Middleware mmc version.

Typedefs

- typedef void(* [mmc_io_strength_t](#))(uint32_t busFreq)
card io strength control

Enumerations

- enum `_mmc_card_flag` {
 - `kMMC_SupportHighSpeed26MHZFlag` = (1U << 0U),
 - `kMMC_SupportHighSpeed52MHZFlag` = (1U << 1U),
 - `kMMC_SupportHighSpeedDDR52MHZ180V300VFlag` = (1 << 2U),
 - `kMMC_SupportHighSpeedDDR52MHZ120VFlag` = (1 << 3U),
 - `kMMC_SupportHS200200MHZ180VFlag` = (1 << 4U),
 - `kMMC_SupportHS200200MHZ120VFlag` = (1 << 5U),
 - `kMMC_SupportHS400DDR200MHZ180VFlag` = (1 << 6U),
 - `kMMC_SupportHS400DDR200MHZ120VFlag` = (1 << 7U),
 - `kMMC_SupportHighCapacityFlag` = (1U << 8U),
 - `kMMC_SupportAlternateBootFlag` = (1U << 9U),
 - `kMMC_SupportDDRBootFlag` = (1U << 10U),
 - `kMMC_SupportHighSpeedBootFlag` = (1U << 11U),
 - `kMMC_SupportEnhanceHS400StrobeFlag` = (1U << 12U) }

MMC card flags.

MMCCARD Function

- `status_t MMC_Init (mmc_card_t *card)`
Initializes the MMC card and host.
- `void MMC_Deinit (mmc_card_t *card)`
Deinitializes the card and host.
- `status_t MMC_CardInit (mmc_card_t *card)`
intialize the card.
- `void MMC_CardDeinit (mmc_card_t *card)`
Deinitializes the card.
- `status_t MMC_HostInit (mmc_card_t *card)`
initialize the host.
- `void MMC_HostDeinit (mmc_card_t *card)`
Deinitializes the host.
- `void MMC_HostReset (SDMMCHOST_CONFIG *host)`
reset the host.
- `void MMC_PowerOnCard (SDMMCHOST_TYPE *base, const sdmmchost_pwr_card_t *pwr)`
power on card.
- `void MMC_PowerOffCard (SDMMCHOST_TYPE *base, const sdmmchost_pwr_card_t *pwr)`
power off card.
- `void MMC_SetCardPower (mmc_card_t *card, bool enable)`
set card power.
- `bool MMC_CheckReadOnly (mmc_card_t *card)`
Checks if the card is read-only.
- `status_t MMC_ReadBlocks (mmc_card_t *card, uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)`
Reads data blocks from the card.
- `status_t MMC_WriteBlocks (mmc_card_t *card, const uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)`

MMC Card Driver

- Writes data blocks to the card.*
- [status_t MMC_EraseGroups](#) ([mmc_card_t](#) *card, [uint32_t](#) startGroup, [uint32_t](#) endGroup)
Erases groups of the card.
- [status_t MMC_SelectPartition](#) ([mmc_card_t](#) *card, [mmc_access_partition_t](#) partitionNumber)
Selects the partition to access.
- [status_t MMC_SetBootConfig](#) ([mmc_card_t](#) *card, const [mmc_boot_config_t](#) *config)
Configures the boot activity of the card.
- [status_t MMC_StartBoot](#) ([mmc_card_t](#) *card, const [mmc_boot_config_t](#) *mmcConfig, [uint8_t](#) *buffer, [sdmmchost_boot_config_t](#) *hostConfig)
MMC card start boot.
- [status_t MMC_SetBootConfigWP](#) ([mmc_card_t](#) *card, [uint8_t](#) wp)
MMC card set boot configuration write protect.
- [status_t MMC_ReadBootData](#) ([mmc_card_t](#) *card, [uint8_t](#) *buffer, [sdmmchost_boot_config_t](#) *hostConfig)
MMC card continous read boot data.
- [status_t MMC_StopBoot](#) ([mmc_card_t](#) *card, [uint32_t](#) bootMode)
MMC card stop boot mode.
- [status_t MMC_SetBootPartitionWP](#) ([mmc_card_t](#) *card, [mmc_boot_partition_wp_t](#) bootPartition-WP)
MMC card set boot partition write protect.

44.4.3 Data Structure Documentation

44.4.3.1 struct mmc_usr_param_t

Data Fields

- [mmc_io_strength_t](#) ioStrength
swith sd io strength
- [uint32_t](#) maxFreq
board support maximum frequency
- [uint32_t](#) capability
board capability flag

44.4.3.2 struct mmc_card_t

Define the card structure including the necessary fields to identify and describe the card.

Data Fields

- [sdmmchost_t](#) * host
Host information.
- [mmc_usr_param_t](#) usrParam
user parameter
- bool isHostReady
Use this flag to indicate if need host re-init or not.
- bool noInternalAlign

- *use this flag to disable sdmmc align.*
- uint32_t [busClock_Hz](#)
MMC bus clock united in Hz.
- uint32_t [relativeAddress](#)
Relative address of the card.
- bool [enablePreDefinedBlockCount](#)
Enable PRE-DEFINED block count when read/write.
- uint32_t [flags](#)
Capability flag in _mmc_card_flag.
- uint8_t [internalBuffer](#) [FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE]
raw buffer used for mmc driver internal
- uint32_t [ocr](#)
Raw OCR content.
- mmc_cid_t [cid](#)
CID.
- mmc_csd_t [csd](#)
CSD.
- mmc_extended_csd_t [extendedCsd](#)
Extended CSD.
- uint32_t [blockSize](#)
Card block size.
- uint32_t [userPartitionBlocks](#)
Card total block number in user partition.
- uint32_t [bootPartitionBlocks](#)
Boot partition size united as block size.
- uint32_t [eraseGroupBlocks](#)
Erase group size united as block size.
- mmc_access_partition_t [currentPartition](#)
Current access partition.
- mmc_voltage_window_t [hostVoltageWindowVCCQ](#)
application must set this value according to board specific
- mmc_voltage_window_t [hostVoltageWindowVCC](#)
application must set this value according to board specific
- mmc_high_speed_timing_t [busTiming](#)
indicate the current work timing mode
- mmc_data_bus_width_t [busWidth](#)
indicate the current work bus width

44.4.3.2.0.47 Field Documentation

44.4.3.2.0.47.1 bool mmc_card_t::noInternalAlign

If disable, sdmmc will not make sure the data buffer address is word align, otherwise all the transfer are align to low level driver

MMC Card Driver

44.4.4 Macro Definition Documentation

44.4.4.1 `#define FSL_MMC_DRIVER_VERSION (MAKE_VERSION(2U, 3U, 0U)) /*2.3.0*/`

44.4.5 Enumeration Type Documentation

44.4.5.1 `enum _mmc_card_flag`

Enumerator

kMMC_SupportHighSpeed26MHZFlag Support high speed 26MHZ.
kMMC_SupportHighSpeed52MHZFlag Support high speed 52MHZ.
kMMC_SupportHighSpeedDDR52MHZ180V300VFlag ddr 52MHZ 1.8V or 3.0V
kMMC_SupportHighSpeedDDR52MHZ120VFlag DDR 52MHZ 1.2V.
kMMC_SupportHS200200MHZ180VFlag HS200, 200MHZ, 1.8V.
kMMC_SupportHS200200MHZ120VFlag HS200, 200MHZ, 1.2V.
kMMC_SupportHS400DDR200MHZ180VFlag HS400, DDR, 200MHZ, 1.8V.
kMMC_SupportHS400DDR200MHZ120VFlag HS400, DDR, 200MHZ, 1.2V.
kMMC_SupportHighCapacityFlag Support high capacity.
kMMC_SupportAlternateBootFlag Support alternate boot.
kMMC_SupportDDRBootFlag support DDR boot flag
kMMC_SupportHighSpeedBootFlag support high speed boot flag
kMMC_SupportEnhanceHS400StrobeFlag support enhance HS400 strobe

44.4.6 Function Documentation

44.4.6.1 `status_t MMC_Init (mmc_card_t * card)`

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_Host-NotReady</i>	host is not ready.
<i>kStatus_SDMMC_Go-IdleFailed</i>	Go idle failed.

<i>kStatus_SDMMC_Send-OperationCondition-Failed</i>	Send operation condition failed.
<i>kStatus_SDMMC_All-SendCidFailed</i>	Send CID failed.
<i>kStatus_SDMMC_Set-RelativeAddressFailed</i>	Set relative address failed.
<i>kStatus_SDMMC_Send-CsdFailed</i>	Send CSD failed.
<i>kStatus_SDMMC_Card-NotSupport</i>	Card not support.
<i>kStatus_SDMMC_Select-CardFailed</i>	Send SELECT_CARD command failed.
<i>kStatus_SDMMC_Send-ExtendedCsdFailed</i>	Send EXT_CSD failed.
<i>kStatus_SDMMC_SetBus-WidthFailed</i>	Set bus width failed.
<i>kStatus_SDMMC_Switch-HighSpeedFailed</i>	Switch high speed failed.
<i>kStatus_SDMMC_Set-CardBlockSizeFailed</i>	Set card block size failed.
<i>kStatus_Success</i>	Operate successfully.

44.4.6.2 void MMC_Deinit (mmc_card_t * *card*)

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

44.4.6.3 status_t MMC_CardInit (mmc_card_t * *card*)

Parameters

MMC Card Driver

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_Host-NotReady</i>	host is not ready.
<i>kStatus_SDMMC_Go-IdleFailed</i>	Go idle failed.
<i>kStatus_SDMMC_Send-OperationCondition-Failed</i>	Send operation condition failed.
<i>kStatus_SDMMC_All-SendCidFailed</i>	Send CID failed.
<i>kStatus_SDMMC_Set-RelativeAddressFailed</i>	Set relative address failed.
<i>kStatus_SDMMC_Send-CsdFailed</i>	Send CSD failed.
<i>kStatus_SDMMC_Card-NotSupport</i>	Card not support.
<i>kStatus_SDMMC_Select-CardFailed</i>	Send SELECT_CARD command failed.
<i>kStatus_SDMMC_Send-ExtendedCsdFailed</i>	Send EXT_CSD failed.
<i>kStatus_SDMMC_SetBus-WidthFailed</i>	Set bus width failed.
<i>kStatus_SDMMC_Switch-HighSpeedFailed</i>	Switch high speed failed.
<i>kStatus_SDMMC_Set-CardBlockSizeFailed</i>	Set card block size failed.
<i>kStatus_Success</i>	Operate successfully.

44.4.6.4 void MMC_CardDeinit (mmc_card_t * *card*)

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

44.4.6.5 **status_t MMC_HostInit (mmc_card_t * *card*)**

This function deinitializes the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

44.4.6.6 **void MMC_HostDeinit (mmc_card_t * *card*)**

This function deinitializes the host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

44.4.6.7 **void MMC_HostReset (SDMMCHOST_CONFIG * *host*)**

This function reset the specific host.

Parameters

<i>host</i>	host descriptor.
-------------	------------------

44.4.6.8 **void MMC_PowerOnCard (SDMMCHOST_TYPE * *base*, const sdmmchost_pwr_card_t * *pwr*)**

The power on operation depend on host or the user define power on function.

Deprecated Do not use this function. It has been superceded by [MMC_SetCardPower](#).

Parameters

MMC Card Driver

<i>base</i>	host base address.
<i>pwr</i>	user define power control configuration

44.4.6.9 void MMC_PowerOffCard (SDMMCHOST_TYPE * *base*, const sdmmchost_pwr_card_t * *pwr*)

The power off operation depend on host or the user define power on function.

Deprecated Do not use this function. It has been superceded by [MMC_SetCardPower](#).

Parameters

<i>base</i>	host base address.
<i>pwr</i>	user define power control configuration

44.4.6.10 void MMC_SetCardPower (mmc_card_t * *card*, bool *enable*)

Parameters

<i>card</i>	card descriptor.
<i>enable</i>	true is power on, false is power off.

44.4.6.11 bool MMC_CheckReadOnly (mmc_card_t * *card*)

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>true</i>	Card is read only.
<i>false</i>	Card isn't read only.

44.4.6.12 status_t MMC_ReadBlocks (mmc_card_t * *card*, uint8_t * *buffer*, uint32_t *startBlock*, uint32_t *blockCount*)

Parameters

<i>card</i>	Card descriptor.
<i>buffer</i>	The buffer to save data.
<i>startBlock</i>	The start block index.
<i>blockCount</i>	The number of blocks to read.

Return values

<i>kStatus_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_Card-NotSupport</i>	Card not support.
<i>kStatus_SDMMC_Set-BlockCountFailed</i>	Set block count failed.
<i>kStatus_SDMMC_-TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_Stop-TransmissionFailed</i>	Stop transmission failed.
<i>kStatus_Success</i>	Operate successfully.

44.4.6.13 **status_t MMC_WriteBlocks (mmc_card_t * *card*, const uint8_t * *buffer*, uint32_t *startBlock*, uint32_t *blockCount*)**

Parameters

<i>card</i>	Card descriptor.
<i>buffer</i>	The buffer to save data blocks.
<i>startBlock</i>	Start block number to write.
<i>blockCount</i>	Block count.

Return values

<i>kStatus_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_Not-SupportYet</i>	Not support now.

MMC Card Driver

<i>kStatus_SDMMC_SetBlockCountFailed</i>	Set block count failed.
<i>kStatus_SDMMC_WaitWriteCompleteFailed</i>	Send status failed.
<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_StopTransmissionFailed</i>	Stop transmission failed.
<i>kStatus_Success</i>	Operate successfully.

44.4.6.14 **status_t MMC_EraseGroups (mmc_card_t * *card*, uint32_t *startGroup*, uint32_t *endGroup*)**

Erase group is the smallest erase unit in MMC card. The erase range is [*startGroup*, *endGroup*].

Parameters

<i>card</i>	Card descriptor.
<i>startGroup</i>	Start group number.
<i>endGroup</i>	End group number.

Return values

<i>kStatus_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_WaitWriteCompleteFailed</i>	Send status failed.
<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_Success</i>	Operate successfully.

44.4.6.15 **status_t MMC_SelectPartition (mmc_card_t * *card*, mmc_access_partition_t *partitionNumber*)**

Parameters

<i>card</i>	Card descriptor.
<i>partition-Number</i>	The partition number.

Return values

<i>kStatus_SDMMC_-ConfigureExtendedCsd-Failed</i>	Configure EXT_CSD failed.
<i>kStatus_Success</i>	Operate successfully.

44.4.6.16 `status_t MMC_SetBootConfig (mmc_card_t * card, const mmc_boot_config_t * config)`

Parameters

<i>card</i>	Card descriptor.
<i>config</i>	Boot configuration structure.

Return values

<i>kStatus_SDMMC_Not-SupportYet</i>	Not support now.
<i>kStatus_SDMMC_-ConfigureExtendedCsd-Failed</i>	Configure EXT_CSD failed.
<i>kStatus_SDMMC_-ConfigureBootFailed</i>	Configure boot failed.
<i>kStatus_Success</i>	Operate successfully.

44.4.6.17 `status_t MMC_StartBoot (mmc_card_t * card, const mmc_boot_config_t * mmcConfig, uint8_t * buffer, sdmmchost_boot_config_t * hostConfig)`

Parameters

MMC Card Driver

<i>card</i>	Card descriptor.
<i>mmcConfig</i>	mmc Boot configuration structure.
<i>buffer</i>	address to recieve data.
<i>hostConfig</i>	host boot configurations.

Return values

<i>kStatus_Fail</i>	fail.
<i>kStatus_SDMMC_-TransferFailed</i>	transfer fail.
<i>kStatus_SDMMC_Go-IdleFailed</i>	reset card fail.
<i>kStatus_Success</i>	Operate successfully.

44.4.6.18 **status_t MMC_SetBootConfigWP (mmc_card_t * *card*, uint8_t *wp*)**

Parameters

<i>card</i>	Card descriptor.
<i>wp</i>	write protect value.

44.4.6.19 **status_t MMC_ReadBootData (mmc_card_t * *card*, uint8_t * *buffer*, sdmmc_host_boot_config_t * *hostConfig*)**

Parameters

<i>card</i>	Card descriptor.
<i>buffer</i>	buffer address.
<i>hostConfig</i>	host boot configurations.

44.4.6.20 **status_t MMC_StopBoot (mmc_card_t * *card*, uint32_t *bootMode*)**

Parameters

<i>card</i>	Card descriptor.
<i>bootMode</i>	boot mode.

44.4.6.21 `status_t MMC_SetBootPartitionWP (mmc_card_t * card,
mmc_boot_partition_wp_t bootPartitionWP)`

Parameters

<i>card</i>	Card descriptor.
<i>bootPartition- WP</i>	boot partition write protect value.



HOST Driver

HOST Driver

44.5.1 Overview

The host adapter driver provide adapter for blocking/non_blocking mode.

Modules

- [SDHC HOST adapter Driver](#)

SDMMC OSA

44.6.1 Overview

The sdmmc osa adapter provide interface of os adapter.

Macros

- #define [SDMMC_OSA_EVENT_TRANSFER_CMD_SUCCESS](#) (1U << 0U)
transfer event
- #define [SDMMC_OSA_EVENT_CARD_INSERTED](#) (1U << 8U)
card detect event, start from index 8

sdmmc osa Function

- void [SDMMC_OSAInit](#) (void)
Initialize OSA.
- [status_t SDMMC_OSAEventCreate](#) (void *eventHandle)
OSA Create event.
- [status_t SDMMC_OSAEventWait](#) (void *eventHandle, uint32_t eventType, uint32_t timeout-Milliseconds, uint32_t *event)
Wait event.
- [status_t SDMMC_OSAEventSet](#) (void *eventHandle, uint32_t eventType)
set event.
- [status_t SDMMC_OSAEventGet](#) (void *eventHandle, uint32_t mask, uint32_t *flag)
Get event flag.
- [status_t SDMMC_OSAEventClear](#) (void *eventHandle, uint32_t eventType)
clear event flag.
- [status_t SDMMC_OSAEventDestroy](#) (void *eventHandle)
Delete event.
- void [SDMMC_OSADelay](#) (uint32_t milliseconds)
sdmmc delay.

44.6.2 Function Documentation

44.6.2.1 [status_t SDMMC_OSAEventCreate](#) (void * *eventHandle*)

Parameters

SDMMC OSA

<i>eventHandle</i>	event handle.
--------------------	---------------

Return values

<i>kStatus_Fail</i>	or kStatus_Success.
---------------------	---------------------

44.6.2.2 `status_t SDMMC_OSAEventWait (void * eventHandle, uint32_t eventType,
uint32_t timeoutMilliseconds, uint32_t * event)`

Parameters

<i>eventHandle</i>	The event type
<i>eventType</i>	Timeout time in milliseconds.
<i>timeout- Milliseconds</i>	timeout value in ms.
<i>event</i>	event flags.

Return values

<i>kStatus_Fail</i>	or kStatus_Success.
---------------------	---------------------

44.6.2.3 `status_t SDMMC_OSAEventSet (void * eventHandle, uint32_t eventType)`

Parameters

<i>eventHandle</i>	event handle.
<i>eventType</i>	The event type

Return values

<i>kStatus_Fail</i>	or kStatus_Success.
---------------------	---------------------

44.6.2.4 `status_t SDMMC_OSAEventGet (void * eventHandle, uint32_t mask, uint32_t *
flag)`

Parameters

<i>eventHandle</i>	event handle.
<i>mask</i>	event mask.
<i>flag</i>	pointer to store event value.

Return values

<i>kStatus_Fail</i>	or kStatus_Success.
---------------------	---------------------

44.6.2.5 status_t SDMMC_OSAEventClear (void * *eventHandle*, uint32_t *eventType*)

Parameters

<i>eventHandle</i>	event handle.
<i>eventType</i>	The event type

Return values

<i>kStatus_Fail</i>	or kStatus_Success.
---------------------	---------------------

44.6.2.6 status_t SDMMC_OSAEventDestroy (void * *eventHandle*)

Parameters

<i>eventHandle</i>	The event handle.
--------------------	-------------------

44.6.2.7 void SDMMC_OSADelay (uint32_t *milliseconds*)

Parameters

<i>milliseconds</i>	time to delay
---------------------	---------------

44.6.3 SDHC HOST adapter Driver

44.6.3.1 Overview

The SDHC host adapter driver provide adapter for blocking/non_blocking mode.

Data Structures

- struct [sdmmchost_pwr_card_t](#)
card power control [More...](#)
- struct [sdmmchost_t](#)
sdmmc host handler [More...](#)

Macros

- #define [FSL_SDMMC_HOST_ADAPTER_VERSION](#) ([MAKE_VERSION](#)(2U, 3U, 0U)) /*2.3.0*/
Middleware adapter version.
- #define [SDMMCHOST_SUPPORT_HIGH_SPEED](#) (1U)
host capability
- #define [SDMMCHOST_DMA_DESCRIPTOR_BUFFER_ALIGN_SIZE](#) (4U)
SDMMC host dma descriptor buffer address align size.

Typedefs

- typedef [sdhc_transfer_t](#) [sdmmchost_transfer_t](#)
sdmmc host transfer function
- typedef void(* [sdmmchost_pwr_t](#))(void)
card power control function pointer

Enumerations

- enum [_sdmmchost_endian_mode](#) {
 [kSDMMCHOST_EndianModeBig](#) = 0U,
 [kSDMMCHOST_EndianModeHalfWordBig](#) = 1U,
 [kSDMMCHOST_EndianModeLittle](#) = 2U }
host Endian mode corresponding to driver define

SDHC host controller function

- void [SDMMCHOST_SetCardBusWidth](#) ([sdmmchost_t](#) *host, uint32_t dataBusWidth)
set data bus width.
- static void [SDMMCHOST_SendCardActive](#) ([sdmmchost_t](#) *host)
Send initialization active 80 clocks to card.
- static uint32_t [SDMMCHOST_SetCardClock](#) ([sdmmchost_t](#) *host, uint32_t targetClock)

- *Set card bus clock.*
- static bool `SDMMCHOST_IsCardBusy (sdmmchost_t *host)`
check card status by DATA0.
- `status_t SDMMCHOST_StartBoot (sdmmchost_t *host, sdmmchost_boot_config_t *hostConfig, sdmmchost_cmd_t *cmd, uint8_t *buffer)`
start read boot data.
- `status_t SDMMCHOST_ReadBootData (sdmmchost_t *host, sdmmchost_boot_config_t *hostConfig, uint8_t *buffer)`
read boot data.
- static void `SDMMCHOST_EnableBoot (sdmmchost_t *host, bool enable)`
enable boot mode.
- static void `SDMMCHOST_EnableCardInt (sdmmchost_t *host, bool enable)`
enable card interrupt.
- `status_t SDMMCHOST_CardIntInit (sdmmchost_t *host, void *sdioInt)`
card interrupt function.
- `status_t SDMMCHOST_CardDetectInit (sdmmchost_t *host, void *cd)`
card detect init function.
- `status_t SDMMCHOST_PollingCardDetectStatus (sdmmchost_t *host, uint32_t waitCardStatus, uint32_t timeout)`
Detect card insert, only need for SD cases.
- `uint32_t SDMMCHOST_CardDetectStatus (sdmmchost_t *host)`
card detect status.
- `status_t SDMMCHOST_Init (sdmmchost_t *host)`
Init host controller.
- void `SDMMCHOST_Deinit (sdmmchost_t *host)`
Deinit host controller.
- void `SDMMCHOST_SetCardPower (sdmmchost_t *host, bool enable)`
host power off card function.
- `status_t SDMMCHOST_TransferFunction (sdmmchost_t *host, sdmmchost_transfer_t *content)`
host transfer function.
- void `SDMMCHOST_Reset (sdmmchost_t *host)`
host reset function.
- `status_t SDMMCHOST_WaitCardDetectStatus (SDMMCHOST_TYPE *hostBase, const sdmmchost-_detect_card_t *cd, bool waitCardStatus)`
wait card detect status
- void `SDMMCHOST_PowerOffCard (SDMMCHOST_TYPE *base, const sdmmchost_pwr_card_t *pwr)`
host power off card function.
- void `SDMMCHOST_PowerOnCard (SDMMCHOST_TYPE *base, const sdmmchost_pwr_card_t *pwr)`
host power on card function.

44.6.3.2 Data Structure Documentation

44.6.3.2.1 struct sdmmchost_pwr_card_t

Deprecated Do not use this structure anymore.

SDMMC OSA

Data Fields

- [sdmmchost_pwr_t powerOn](#)
power on function pointer
- [uint32_t powerOnDelay_ms](#)
power on delay
- [sdmmchost_pwr_t powerOff](#)
power off function pointer
- [uint32_t powerOffDelay_ms](#)
power off delay

44.6.3.2.2 struct sdmmchost_t

Data Fields

- [sdhc_host_t hostController](#)
host configuration
- [void * dmaDesBuffer](#)
DMA descriptor buffer address.
- [uint32_t dmaDesBufferWordsNum](#)
DMA descriptor buffer size in byte.
- [sdhc_handle_t handle](#)
host controller handler
- [void * hostEvent](#)
host event handler pointer
- [void * cd](#)
card detect
- [void * cardInt](#)
call back function for card interrupt

44.6.3.3 Macro Definition Documentation

44.6.3.3.1 **#define FSL_SDMMC_HOST_ADAPTER_VERSION (MAKE_VERSION(2U, 3U, 0U))**
*/*2.3.0*/*

44.6.3.4 Enumeration Type Documentation

44.6.3.4.1 enum _sdmmchost_endian_mode

Enumerator

kSDMMCHOST_EndianModeBig Big endian mode.

kSDMMCHOST_EndianModeHalfWordBig Half word big endian mode.

kSDMMCHOST_EndianModeLittle Little endian mode.

44.6.3.5 Function Documentation

44.6.3.5.1 void SDMMCHOST_SetCardBusWidth (sdmmchost_t * *host*, uint32_t *dataBusWidth*)

SDMMC OSA

Parameters

<i>host</i>	host handler
<i>dataBusWidth</i>	data bus width

44.6.3.5.2 `static void SDMMCHOST_SendCardActive (sdmmchost_t * host) [inline], [static]`

Parameters

<i>host</i>	host handler
-------------	--------------

44.6.3.5.3 `static uint32_t SDMMCHOST_SetCardClock (sdmmchost_t * host, uint32_t targetClock) [inline], [static]`

Parameters

<i>host</i>	host handler
<i>targetClock</i>	target clock frequency

Return values

<i>actual</i>	clock frequency can be reach.
---------------	-------------------------------

44.6.3.5.4 `static bool SDMMCHOST_IsCardBusy (sdmmchost_t * host) [inline], [static]`

Parameters

<i>host</i>	host handler
-------------	--------------

Return values

<i>true</i>	is busy, false is idle.
-------------	-------------------------

44.6.3.5.5 `status_t SDMMCHOST_StartBoot (sdmmchost_t * host, sdmmchost_boot_config_t * hostConfig, sdmmchost_cmd_t * cmd, uint8_t * buffer)`

Parameters

<i>host</i>	host handler
<i>hostConfig</i>	boot configuration
<i>cmd</i>	boot command
<i>buffer</i>	buffer address

44.6.3.5.6 `status_t SDMMCHOST_ReadBootData (sdmmchost_t * host,
sdmmchost_boot_config_t * hostConfig, uint8_t * buffer)`

Parameters

<i>host</i>	host handler
<i>hostConfig</i>	boot configuration
<i>buffer</i>	buffer address

44.6.3.5.7 `static void SDMMCHOST_EnableBoot (sdmmchost_t * host, bool enable)
[inline], [static]`

Parameters

<i>host</i>	host handler
<i>enable</i>	true is enable, false is disable

44.6.3.5.8 `static void SDMMCHOST_EnableCardInt (sdmmchost_t * host, bool enable)
[inline], [static]`

Parameters

<i>host</i>	host handler
<i>enable</i>	true is enable, false is disable.

44.6.3.5.9 `status_t SDMMCHOST_CardIntInit (sdmmchost_t * host, void * sdioInt)`

SDMMC OSA

Parameters

<i>host</i>	host handler
<i>sdioInt</i>	card interrupt configuration

44.6.3.5.10 **status_t SDMMCHOST_CardDetectInit (sdmmchost_t * *host*, void * *cd*)**

Parameters

<i>host</i>	host handler
<i>cd</i>	card detect configuration

44.6.3.5.11 **status_t SDMMCHOST_PollingCardDetectStatus (sdmmchost_t * *host*, uint32_t *waitCardStatus*, uint32_t *timeout*)**

Parameters

<i>host</i>	host handler
<i>waitCardStatus</i>	status which user want to wait
<i>timeout</i>	wait time out.

Return values

<i>kStatus_Success</i>	detect card insert
<i>kStatus_Fail</i>	card insert event fail

44.6.3.5.12 **uint32_t SDMMCHOST_CardDetectStatus (sdmmchost_t * *host*)**

Parameters

<i>host</i>	host handler
-------------	--------------

Return values

<i>kSD_Inserted, kSD_-Removed</i>

44.6.3.5.13 status_t SDMMCHOST_Init (sdmmchost_t * *host*)

Parameters

<i>host</i>	host handler
-------------	--------------

Return values

<i>kStatus_Success</i>	host init success
<i>kStatus_Fail</i>	event fail

44.6.3.5.14 void SDMMCHOST_Deinit (sdmmchost_t * *host*)

Parameters

<i>host</i>	host handler
-------------	--------------

44.6.3.5.15 void SDMMCHOST_SetCardPower (sdmmchost_t * *host*, bool *enable*)

Parameters

<i>host</i>	host handler
<i>enable</i>	true is power on, false is power down.

44.6.3.5.16 status_t SDMMCHOST_TransferFunction (sdmmchost_t * *host*, sdmmchost_transfer_t * *content*)

Parameters

<i>host</i>	host handler
-------------	--------------

SDMMC OSA

<i>content</i>	transfer content.
----------------	-------------------

44.6.3.5.17 void SDMMCHOST_Reset (sdmmchost_t * *host*)

Deprecated Do not use this function. Application should not call this function, driver is responsible for the host reset..

Parameters

<i>host</i>	host handler
-------------	--------------

44.6.3.5.18 status_t SDMMCHOST_WaitCardDetectStatus (SDMMCHOST_TYPE * *hostBase*, const sdmmchost_detect_card_t * *cd*, bool *waitCardStatus*)

Deprecated Do not use this function.It has been superceded by [SDMMCHOST_PollingCardDetect-Status..](#)

Parameters

<i>hostBase</i>	host handler
<i>cd</i>	card detect configuration.
<i>waitCardStatus</i>	status to wait.

44.6.3.5.19 void SDMMCHOST_PowerOffCard (SDMMCHOST_TYPE * *base*, const sdmmchost_pwr_card_t * *pwr*)

Deprecated Do not use this function.It has been superceded by [SDMMCHOST_SetCardPower..](#)

Parameters

<i>base</i>	host base address.
<i>pwr</i>	depend on user define power configuration.

44.6.3.5.20 void SDMMCHOST_PowerOnCard (SDMMCHOST_TYPE * *base*, const sdmmchost_pwr_card_t * *pwr*)

Deprecated Do not use this function.It has been superceded by [SDMMCHOST_SetCardPower..](#)

Parameters

<i>base</i>	host base address.
<i>pwr</i>	depend on user define power configuration.

SDMMC Common

44.7.1 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_BYTE_SEQUENCE\(x\) \(__REV\(x\)\)](#)
Reverse byte sequence in uint32_t.
- #define [SWAP_HALF_WROD_BYTE_SEQUENCE\(x\) \(__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) (1U << (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.

SDMMC Common

- #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_FUNCNCE](#) (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 register IO NUMBER mask*
- #define **SDIO_CCCR_SUPPORT_HIGHSPEED** (1U << 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** (1U << 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 FREQUENCY 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 field 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_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 field 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** (2U)
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 aolocation 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 `_sdmmc_status` {
 - `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_SDMMC,


```

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) }

```

SD/MMC card API's running status.

- enum `_sdmmc_signal_line` {
 - `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 `_sdmmc_bus_width` {
 - `kSDMMC_BusWidth1Bit` = 0U,
 - `kSDMMC_BusWidth4Bit` = 1U,
 - `kSDMMC_BusWidth8Bit` = 2U }

card bus width
- enum `_sdmmc_capability_flag` { `kSDMMC_Support8BitWidth` = 1U }

sdmmc capability flag

- enum `sd_detect_card_type_t` {
 - `kSD_DetectCardByGpioCD`,
 - `kSD_DetectCardByHostCD`,
 - `kSD_DetectCardByHostDATA3` }

sd card detect type
- enum `_sd_card_cd_status` {
 - `kSD_Inserted` = 1U,
 - `kSD_Removed` = 0U }

@ 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 `_sdmmc_r1_card_status_flag` {

SDMMC Common

```
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 }

CURRENT_STATE field in R1.

- enum `_sdspi_r1_error_status_flag` {
 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 `_sdspi_r2_error_status_flag` {

```

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 `_sdspi_data_error_token` {


```

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_ApplicationSetBusWdith = 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 `_sdmmc_command_class` {
 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 `_sd_ocr_flag` {
 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 `_sd_specification_version` {
 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 `_sd_csd_flag` {

```

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 `_sd_scr_flag` {
`kSD_ScrDataStatusAfterErase` = (1U << 0U),
`kSD_ScrSdSpecification3` = (1U << 1U) }

SD card SCR register flags.

- enum `_sd_timing_function` {
`kSD_FunctionSDR12Deafult` = 0U,
`kSD_FunctionSDR25HighSpeed` = 1U,
`kSD_FunctionSDR50` = 2U,
`kSD_FunctionSDR104` = 3U,
`kSD_FunctionDDR50` = 4U }

SD timing function number.

- enum `_sd_group_num` {
`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,

SDMMC Common

`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 _sdio_cccr_reg {`

```

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 _sdio_status_flag {

```

```
kSDIO_StatusCmdCRCError = 0x8000U,
kSDIO_StatusIllegalCmd = 0x4000U,
kSDIO_StatusR6Error = 0x2000U,
kSDIO_StatusError = 0x0800U,
kSDIO_StatusFunctionNumError = 0x0200U,
kSDIO_StatusOutOfRange = 0x0100U }
```

sdio command response flag

- enum `_sdio_ocr_flag` {


```
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 `_sdio_capability_flag` {


```
kSDIO_CCCRSupportDirectCmdDuringDataTrans = (1U << 0U),
kSDIO_CCCRSupportMultiBlock = (1U << 1U),
kSDIO_CCCRSupportReadWait = (1U << 2U),
kSDIO_CCCRSupportSuspendResume = (1U << 3U),
kSDIO_CCCRSupportIntDuring4BitDataTrans = (1U << 4U),
kSDIO_CCCRSupportLowSpeed1Bit = (1U << 6U),
kSDIO_CCCRSupportLowSpeed4Bit = (1U << 7U),
kSDIO_CCCRSupportMasterPowerControl = (1U << 8U),
kSDIO_CCCRSupportHighSpeed = (1U << 9U),
kSDIO_CCCRSupportContinuousSPIInt = (1U << 10U) }
```

sdio capability flag

- enum `_sdio_fbr_flag` {


```
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` }
- MMC card classified as density level.*
- 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 `_mmc_extended_csd_revision` {  
`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 `_mmc_support_boot_mode` {  
`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 mddle intended for both boot areas will be set with a single write.*

- enum `_mmc_boot_partition_wp_status` {  
`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 `_mmc_csd_flag` {

```

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_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 `_mmc_driver_strength` {  
`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) }

*mmc extended csd flags*

- enum `mmc_boot_mode_t` {

```
kMMC_BootModeNormal = 0U,
kMMC_BootModeAlternative = 1U }
MMC card boot mode.
```

## common function

### tuning pattern

- [status\\_t SDMMC\\_SelectCard](#) ([sdmmcchost\\_t](#) \*host, [uint32\\_t](#) relativeAddress, [bool](#) isSelected)  
*Selects the card to put it into transfer state.*
- [status\\_t SDMMC\\_SendApplicationCommand](#) ([sdmmcchost\\_t](#) \*host, [uint32\\_t](#) relativeAddress)  
*Sends an application command.*
- [status\\_t SDMMC\\_SetBlockCount](#) ([sdmmcchost\\_t](#) \*host, [uint32\\_t](#) blockCount)  
*Sets the block count.*
- [status\\_t SDMMC\\_GoIdle](#) ([sdmmcchost\\_t](#) \*host)  
*Sets the card to be idle state.*
- [status\\_t SDMMC\\_SetBlockSize](#) ([sdmmcchost\\_t](#) \*host, [uint32\\_t](#) blockSize)  
*Sets data block size.*
- [status\\_t SDMMC\\_SetCardInactive](#) ([sdmmcchost\\_t](#) \*host)  
*Sets card to inactive status.*

## 44.7.2 Data Structure Documentation

### 44.7.2.1 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*
- [void \\*](#) [userData](#)  
*user data*

### 44.7.2.2 struct sd\_io\_voltage\_t

#### Data Fields

- [sd\\_io\\_voltage\\_ctrl\\_type\\_t](#) type  
*io voltage switch type*
- [sd\\_io\\_voltage\\_func\\_t](#) [func](#)  
*io voltage switch function*

### 44.7.2.3 struct sd\_usr\_param\_t

#### Data Fields

- [sd\\_pwr\\_t](#) pwr  
*power control configuration pointer*
- [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*

### 44.7.2.4 struct sdio\_card\_int\_t

#### Data Fields

- void \* userData  
*user data*
- [sdio\\_int\\_t](#) cardInterrupt  
*card int call back*

### 44.7.2.5 struct sdio\_usr\_param\_t

#### Data Fields

- [sd\\_pwr\\_t](#) pwr  
*power control configuration pointer*
- [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*

#### 44.7.2.6 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*

#### 44.7.2.7 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*

#### 44.7.2.8 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](#)

## SDMMC Common

- *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*

### 44.7.2.9 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*

#### 44.7.2.10 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].*

#### 44.7.2.11 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](#)  
*Data read access-time-2 in clock cycles (NSAC\*100) [111:104].*
- 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](#)

## SDMMC Common

- *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].*

### 44.7.2.12 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]*

### 44.7.2.13 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.*

#### 44.7.2.14 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](#)  
*Max.*
- 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](#)

## SDMMC Common

- *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].*

### 44.7.2.14.0.1 Field Documentation

#### 44.7.2.14.0.1.1 uint8\_t mmc\_csd\_t::transferSpeed

bus clock frequency [103:96]

#### 44.7.2.14.0.1.2 uint8\_t mmc\_csd\_t::readBlockLength

read data block length [83:80]

#### 44.7.2.14.0.1.3 uint8\_t mmc\_csd\_t::readCurrentVddMin

read current @ VDD min [61:59]

#### 44.7.2.14.0.1.4 uint8\_t mmc\_csd\_t::readCurrentVddMax

read current @ VDD max [58:56]

#### 44.7.2.14.0.1.5 uint8\_t mmc\_csd\_t::writeCurrentVddMin

write current @ VDD min [55:53]

#### 44.7.2.14.0.1.6 uint8\_t mmc\_csd\_t::writeCurrentVddMax

write current @ VDD max [52:50]

#### 44.7.2.14.0.1.7 uint8\_t mmc\_csd\_t::maxWriteBlockLength

write data block length [25:22]

### 44.7.2.15 struct mmc\_extended\_csd\_t

#### Data Fields

- uint8\_t [partitionAttribute](#)  
*< secure removal type [16]*
- 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](#)  
*Erased 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 [extendedCsdVersion](#)  
*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 [powerClass52MHz195V](#)  
*< out of interrupt busy timing [198]*
- 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](#)

## SDMMC Common

- *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 cacheSize`  
    *< 1st initialization time after partitioning[241]*
- `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]*

### 44.7.2.15.0.2 Field Documentation

#### 44.7.2.15.0.2.1 `uint8_t mmc_extended_csd_t::partitionAttribute`

- < 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]
- < power off notification[34]
- < 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]

#### **44.7.2.15.0.2.2 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]

#### **44.7.2.15.0.2.3 uint8\_t mmc\_extended\_csd\_t::powerClass52MHz195V**

- < partition switch timing [199] Power Class for 52MHz @ 1.95V [200]

#### **44.7.2.15.0.2.4 uint8\_t mmc\_extended\_csd\_t::sleepAwakeTimeout**

Sleep/awake timeout [217]

## SDMMC Common

### 44.7.2.15.0.2.5 uint8\_t mmc\_extended\_csd\_t::sleepCurrentVCCQ

Sleep current (VCCQ) [219]

### 44.7.2.15.0.2.6 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]

### 44.7.2.15.0.2.7 uint32\_t mmc\_extended\_csd\_t::cacheSize

< correct prg sectors number[245-242]

< background operations status[246]

< power off notification timeout[247]

< generic CMD6 timeout[248] cache size[252-249]

### 44.7.2.15.0.2.8 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]

### 44.7.2.15.0.2.9 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]



#### 44.7.2.16 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.*

#### 44.7.2.17 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*

### 44.7.3 Macro Definition Documentation

44.7.3.1 **#define SDMMC\_LOG( *format*, ... )**

44.7.3.2 **#define READ\_MMC\_TRANSFER\_SPEED\_FREQUENCY\_UNIT( *CSD* ) (((CSD.-transferSpeed) & MMC\_TRANSFER\_SPEED\_FREQUENCY\_UNIT\_MASK) >> MMC\_TRANSFER\_SPEED\_FREQUENCY\_UNIT\_SHIFT)**

44.7.3.3 **#define READ\_MMC\_TRANSFER\_SPEED\_MULTIPLIER( *CSD* ) (((CSD.transferSpeed) & MMC\_TRANSFER\_SPEED\_MULTIPLIER\_MASK) >> MMC\_TRANSFER\_SPEED\_MULTIPLIER\_SHIFT)**

44.7.3.4 **#define MMC\_EXTENDED\_CSD\_BYTES (512U)**

44.7.3.5 **#define SD\_PRODUCT\_NAME\_BYTES (5U)**

44.7.3.6 **#define MMC\_PRODUCT\_NAME\_BYTES (6U)**

44.7.3.7 **#define MMC\_SWITCH\_COMMAND\_SET\_SHIFT (0U)**

44.7.3.8 **#define MMC\_SWITCH\_COMMAND\_SET\_MASK (0x00000007U)**

### 44.7.4 Enumeration Type Documentation

44.7.4.1 **enum \_sdmmc\_status**

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.

#### 44.7.4.2 enum\_sdmmc\_signal\_line

Enumerator

*kSDMMC\_SignalLineCmd* cmd line  
*kSDMMC\_SignalLineData0* data line  
*kSDMMC\_SignalLineData1* data line  
*kSDMMC\_SignalLineData2* data line  
*kSDMMC\_SignalLineData3* data line  
*kSDMMC\_SignalLineData4* data line  
*kSDMMC\_SignalLineData5* data line  
*kSDMMC\_SignalLineData6* data line  
*kSDMMC\_SignalLineData7* data line

## SDMMC Common

### 44.7.4.3 enum sdmmc\_operation\_voltage\_t

Enumerator

|                                    |                                                          |
|------------------------------------|----------------------------------------------------------|
| <i>kSDMMC_OperationVoltageNone</i> | indicate current voltage setting is not setting by suser |
| <i>kSDMMC_OperationVoltage330V</i> | card operation voltage around 3.3v                       |
| <i>kSDMMC_OperationVoltage300V</i> | card operation voltage around 3.0v                       |
| <i>kSDMMC_OperationVoltage180V</i> | card operation voltage around 1.8v                       |

### 44.7.4.4 enum \_sdmmc\_bus\_width

Enumerator

|                            |                  |
|----------------------------|------------------|
| <i>kSDMMC_BusWdith1Bit</i> | card bus 1 width |
| <i>kSDMMC_BusWdith4Bit</i> | card bus 4 width |
| <i>kSDMMC_BusWdith8Bit</i> | card bus 8 width |

### 44.7.4.5 enum \_sdmmc\_capability\_flag

Enumerator

|                                |                             |
|--------------------------------|-----------------------------|
| <i>kSDMMC_Support8BitWidth</i> | 8 bit data width capability |
|--------------------------------|-----------------------------|

### 44.7.4.6 enum sd\_detect\_card\_type\_t

Enumerator

|                                  |                                         |
|----------------------------------|-----------------------------------------|
| <i>kSD_DetectCardByGpioCD</i>    | sd card detect by CD pin through GPIO   |
| <i>kSD_DetectCardByHostCD</i>    | sd card detect by CD pin through host   |
| <i>kSD_DetectCardByHostDATA3</i> | sd card detect by DAT3 pin through host |

### 44.7.4.7 enum \_sd\_card\_cd\_status

Enumerator

|                     |                  |
|---------------------|------------------|
| <i>kSD_Inserted</i> | card is inserted |
| <i>kSD_Removed</i>  | card is removed  |

#### 44.7.4.8 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

#### 44.7.4.9 enum \_sdmmc\_r1\_card\_status\_flag

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

#### 44.7.4.10 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.

## SDMMC Common

***kSDMMC\_R1StateReceiveData*** R1: current state: receiving data.

***kSDMMC\_R1StateProgram*** R1: current state: programming.

***kSDMMC\_R1StateDisconnect*** R1: current state: disconnect.

### 44.7.4.11 enum \_sdspi\_r1\_error\_status\_flag

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.

### 44.7.4.12 enum \_sdspi\_r2\_error\_status\_flag

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.

### 44.7.4.13 enum \_sdspi\_data\_error\_token

Enumerator

***kSDSPI\_DataErrorTokenError*** Data error.

***kSDSPI\_DataErrorTokenCardControllerError*** Card controller error.

***kSDSPI\_DataErrorTokenCardEccFailed*** Card ecc error.

***kSDSPI\_DataErrorTokenOutOfRange*** Out of range.

**44.7.4.14 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.**44.7.4.15 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.**44.7.4.16 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.**44.7.4.17 enum sdspi\_command\_t**

Enumerator

*kSDSPI\_CommandCrc* Command crc protection on/off.**44.7.4.18 enum sd\_application\_command\_t**

Enumerator

*kSD\_ApplicationSetBusWidth* Set Bus Width.*kSD\_ApplicationStatus* Send SD status.*kSD\_ApplicationSendNumberWriteBlocks* Send Number Of Written Blocks.

## SDMMC Common

*kSD\_ApplicationSetWriteBlockEraseCount* Set Write Block Erase Count.  
*kSD\_ApplicationSendOperationCondition* Send Operation Condition.  
*kSD\_ApplicationSetClearCardDetect* Set Connect/Disconnect pull up on detect pin.  
*kSD\_ApplicationSendScr* Send Scr.

### 44.7.4.19 enum \_sdmmc\_command\_class

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.

### 44.7.4.20 enum \_sd\_ocr\_flag

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.

### 44.7.4.21 enum \_sd\_specification\_version

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.

#### 44.7.4.22 enum sd\_switch\_mode\_t

Enumerator

*kSD\_SwitchCheck* SD switch mode 0: check function.

*kSD\_SwitchSet* SD switch mode 1: set function.

#### 44.7.4.23 enum \_sd\_csd\_flag

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].

#### 44.7.4.24 enum \_sd\_scr\_flag

Enumerator

*kSD\_ScrDataStatusAfterErase* Data status after erases [55:55].

*kSD\_ScrSdSpecification3* Specification version 3.00 or higher [47:47].

#### 44.7.4.25 enum \_sd\_timing\_function

Enumerator

*kSD\_FunctionSDR12Deafult* SDR12 mode & default.

*kSD\_FunctionSDR25HighSpeed* SDR25 & high speed.

*kSD\_FunctionSDR50* SDR50 mode.

*kSD\_FunctionSDR104* SDR104 mode.

*kSD\_FunctionDDR50* DDR50 mode.

### 44.7.4.26 enum \_sd\_group\_num

Enumerator

*kSD\_GroupTimingMode* access mode group  
*kSD\_GroupCommandSystem* command system group  
*kSD\_GroupDriverStrength* driver strength group  
*kSD\_GroupCurrentLimit* current limit group

### 44.7.4.27 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.

### 44.7.4.28 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

### 44.7.4.29 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

### 44.7.4.30 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.

#### 44.7.4.31 enum \_sdio\_cccr\_reg

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

## SDMMC Common

*kSDIO\_RegBusSpeed* bus speed register  
*kSDIO\_RegUHSITimingSupport* UHS-I timing support register.  
*kSDIO\_RegDriverStrength* Driver strength register.  
*kSDIO\_RegInterruptExtension* Interrupt extension register.

### 44.7.4.32 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

### 44.7.4.33 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

### 44.7.4.34 enum \_sdio\_status\_flag

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

**44.7.4.35 enum \_sdio\_ocr\_flag**

Enumerator

*kSDIO\_OcrPowerUpBusyFlag* Power up busy status.  
*kSDIO\_OcrIONumber* number of IO function  
*kSDIO\_OcrMemPresent* memory present flag  
*kSDIO\_OcrVdd20\_21Flag* VDD 2.0-2.1.  
*kSDIO\_OcrVdd21\_22Flag* VDD 2.1-2.2.  
*kSDIO\_OcrVdd22\_23Flag* VDD 2.2-2.3.  
*kSDIO\_OcrVdd23\_24Flag* VDD 2.3-2.4.  
*kSDIO\_OcrVdd24\_25Flag* VDD 2.4-2.5.  
*kSDIO\_OcrVdd25\_26Flag* VDD 2.5-2.6.  
*kSDIO\_OcrVdd26\_27Flag* VDD 2.6-2.7.  
*kSDIO\_OcrVdd27\_28Flag* VDD 2.7-2.8.  
*kSDIO\_OcrVdd28\_29Flag* VDD 2.8-2.9.  
*kSDIO\_OcrVdd29\_30Flag* VDD 2.9-3.0.  
*kSDIO\_OcrVdd30\_31Flag* VDD 2.9-3.0.  
*kSDIO\_OcrVdd31\_32Flag* VDD 3.0-3.1.  
*kSDIO\_OcrVdd32\_33Flag* VDD 3.1-3.2.  
*kSDIO\_OcrVdd33\_34Flag* VDD 3.2-3.3.  
*kSDIO\_OcrVdd34\_35Flag* VDD 3.3-3.4.  
*kSDIO\_OcrVdd35\_36Flag* VDD 3.4-3.5.

**44.7.4.36 enum \_sdio\_capability\_flag**

Enumerator

*kSDIO\_CCCRSupportDirectCmdDuringDataTrans* support direct cmd during data transfer  
*kSDIO\_CCCRSupportMultiBlock* support multi block mode  
*kSDIO\_CCCRSupportReadWait* support read wait  
*kSDIO\_CCCRSupportSuspendResume* support suspend resume  
*kSDIO\_CCCRSupportIntDuring4BitDataTrans* support interrupt during 4-bit data transfer  
*kSDIO\_CCCRSupportLowSpeed1Bit* support low speed 1bit mode  
*kSDIO\_CCCRSupportLowSpeed4Bit* support low speed 4bit mode  
*kSDIO\_CCCRSupportMasterPowerControl* support master power control  
*kSDIO\_CCCRSupportHighSpeed* support high speed  
*kSDIO\_CCCRSupportContinuousSPIInt* support continuous SPI interrupt

**44.7.4.37 enum \_sdio\_fbr\_flag**

Enumerator

*kSDIO\_FBRSupportCSA* function support CSA

## SDMMC Common

***kSDIO\_FBRSupportPowerSelection*** function support power selection

### 44.7.4.38 enum sdio\_bus\_width\_t

Enumerator

***kSDIO\_DataBus1Bit*** 1 bit bus mode

***kSDIO\_DataBus4Bit*** 4 bit bus mode

***kSDIO\_DataBus8Bit*** 8 bit bus mode

### 44.7.4.39 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.

### 44.7.4.40 enum mmc\_classified\_voltage\_t

Enumerator

***kMMC\_ClassifiedVoltageHigh*** High-voltage MMC card.

***kMMC\_ClassifiedVoltageDual*** Dual-voltage MMC card.

### 44.7.4.41 enum mmc\_classified\_density\_t

Enumerator

***kMMC\_ClassifiedDensityWithin2GB*** Density byte is less than or equal 2GB.

**44.7.4.42 enum mmc\_access\_mode\_t**

Enumerator

- kMMC\_AccessModeByte* The card should be accessed as byte.  
*kMMC\_AccessModeSector* The card should be accessed as sector.

**44.7.4.43 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.

**44.7.4.44 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.

**44.7.4.45 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.

**44.7.4.46 enum \_mmc\_extended\_csd\_revision**

Enumerator

- kMMC\_ExtendedCsdRevision10* Revision 1.0.  
*kMMC\_ExtendedCsdRevision11* Revision 1.1.  
*kMMC\_ExtendedCsdRevision12* Revision 1.2.  
*kMMC\_ExtendedCsdRevision13* Revision 1.3 MMC4.3.

## SDMMC Common

***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.

### 44.7.4.47 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.

### 44.7.4.48 enum \_mmc\_support\_boot\_mode

Enumerator

***kMMC\_SupportAlternateBoot*** support alternative boot mode  
***kMMC\_SupportDDRBoot*** support DDR boot mode  
***kMMC\_SupportHighSpeedBoot*** support high speed boot mode

### 44.7.4.49 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.

### 44.7.4.50 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.



**44.7.4.51 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.

**44.7.4.52 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 data rate

**44.7.4.53 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

**44.7.4.54 enum \_mmc\_boot\_partition\_wp\_status**

Enumerator

*kMMC\_BootPartitionNotProtected* boot partition not protected  
*kMMC\_BootPartitionPwrProtected* boot partition is power on period write protected  
*kMMC\_BootPartitionPermProtected* boot partition is permanently protected

### 44.7.4.55 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

### 44.7.4.56 enum \_mmc\_csd\_flag

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.

### 44.7.4.57 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.

### 44.7.4.58 enum mmc\_extended\_csd\_index\_t

Enumerator

***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.

#### 44.7.4.59 enum \_mmc\_driver\_strength

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.

#### 44.7.4.60 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]

#### 44.7.4.61 enum mmc\_boot\_mode\_t

Enumerator

*kMMC\_BootModeNormal* Normal boot.  
*kMMC\_BootModeAlternative* Alternative boot.

### 44.7.5 Function Documentation

**44.7.5.1** `status_t SDMMC_SelectCard ( sdmmc_host_t * host, uint32_t relativeAddress, bool isSelected )`

## SDMMC Common

### 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. |

#### 44.7.5.2 **status\_t SDMMC\_SendApplicationCommand ( sdmmc\_host\_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. |

#### 44.7.5.3 **status\_t SDMMC\_SetBlockCount ( sdmmc\_host\_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. |

#### 44.7.5.4 status\_t SDMMC\_Goldle ( sdmmc\_host\_t \* host )

Parameters

|             |               |
|-------------|---------------|
| <i>host</i> | host handler. |
|-------------|---------------|

Return values

|                                      |                       |
|--------------------------------------|-----------------------|
| <i>kStatus_SDMMC_-TransferFailed</i> | Transfer failed.      |
| <i>kStatus_Success</i>               | Operate successfully. |

#### 44.7.5.5 status\_t SDMMC\_SetBlockSize ( sdmmc\_host\_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. |

#### 44.7.5.6 status\_t SDMMC\_SetCardInactive ( sdmmc\_host\_t \* host )

Parameters

\_\_\_\_\_

## SDMMC Common

|             |               |
|-------------|---------------|
| <i>host</i> | host handler. |
|-------------|---------------|

Return values

|                                           |                       |
|-------------------------------------------|-----------------------|
| <i>kStatus_SDMMC_-<br/>TransferFailed</i> | Transfer failed.      |
| <i>kStatus_Success</i>                    | Operate successfully. |

## Chapter 45

# SPI based Secure Digital Card (SDSPI)

### 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, 1U, 4U)`) /\*2.1.4\*/  
*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 `_sdspi_status` {  
    `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 `_sdspi_card_flag` {  
    `kSDSPI_SupportHighCapacityFlag` = (1U << 0U),  
    `kSDSPI_SupportSdhcFlag` = (1U << 1U),  
    `kSDSPI_SupportSdxcFlag` = (1U << 2U),  
    `kSDSPI_SupportSdscFlag` = (1U << 3U) }  
*SDSPI card flag.*
- enum `_sdspi_response_type` {  
    `kSDSPI_ResponseTypeR1` = 0U,  
    `kSDSPI_ResponseTypeR1b` = 1U,  
    `kSDSPI_ResponseTypeR2` = 2U,  
    `kSDSPI_ResponseTypeR3` = 3U,  
    `kSDSPI_ResponseTypeR7` = 4U }



- SDSPI response type.*
  - enum `_sdspi_cmd` {
    - `kSDSPI_CmdGoIdle` = `kSDMMC_GoIdleState` << 8U | `kSDSPI_ResponseTypeR1`,
    - `kSDSPI_CmdCrc` = `kSDSPI_CommandCrc` << 8U | `kSDSPI_ResponseTypeR1`,
    - `kSDSPI_CmdSendInterfaceCondition` }
- SDSPI command type.*

## 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_SendCid (sdspi_card_t *card)`  
*Send GET-CID command In our sdspi init function, this function is removed for better code size, if id 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.*

## Data Structure Documentation

### 45.2.1 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.*

### 45.2.2 struct sdspi\_card\_t

Define the card structure including the necessary fields to identify and describe the card.

## Macro Definition Documentation

### 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 rawCid [16U]`  
*Raw CID content.*
- `uint8_t rawCsd [16U]`  
*Raw CSD content.*
- `uint8_t rawScr [8U]`  
*Raw SCR 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.*

#### 45.2.2.0.0.3 Field Documentation

##### 45.2.2.0.0.3.1 `uint32_t sdsapi_card_t::flags`

### Macro Definition Documentation

**45.3.1 `#define FSL_SDSPIC_DRIVER_VERSION (MAKE_VERSION(2U, 1U, 4U))`**  
*/\*2.1.4\*/*

**45.3.2 `#define DSPI_DUMMY_DATA (0xFFU)`**

Dummy data used for Tx if there is no txData.

**45.3.3 `#define SDSPI_CARD_CRC_PROTECTION_ENABLE 0U`**

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.

## Enumeration Type Documentation

### 45.4.1 enum \_sdspi\_status

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

### 45.4.2 enum \_sdspi\_card\_flag

Enumerator

*kSDSPI\_SupportHighCapacityFlag* Card is high capacity.  
*kSDSPI\_SupportSdhcFlag* Card is SDHC.  
*kSDSPI\_SupportSdxcFlag* Card is SDXC.  
*kSDSPI\_SupportSdscFlag* Card is SDSC.

### 45.4.3 enum \_sdspi\_response\_type

Enumerator

*kSDSPI\_ResponseTypeR1* Response 1.

## Function Documentation

***kSDSPI\_ResponseTypeR1b*** Response 1 with busy.

***kSDSPI\_ResponseTypeR2*** Response 2.

***kSDSPI\_ResponseTypeR3*** Response 3.

***kSDSPI\_ResponseTypeR7*** Response 7.

### 45.4.4 enum \_sdspi\_cmd

Enumerator

***kSDSPI\_CmdGoIdle*** command go idle

***kSDSPI\_CmdCrc*** command crc protection

***kSDSPI\_CmdSendInterfaceCondition*** command send interface condition

## Function Documentation

### 45.5.1 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_GoIdle-Failed</i>                  | Go idle failed.                  |
| <i>kStatus_SDSPI_Send-InterfaceConditionFailed</i>  | Send interface condition failed. |
| <i>kStatus_SDSPI_Send-OperationCondition-Failed</i> | Send operation condition failed. |

|                                          |                        |
|------------------------------------------|------------------------|
| <i>kStatus_Timeout</i>                   | Send command timeout.  |
| <i>kStatus_SDSPI_Not-SupportYet</i>      | Not support yet.       |
| <i>kStatus_SDSPI_ReadOcr-Failed</i>      | Read OCR failed.       |
| <i>kStatus_SDSPI_SetBlock-SizeFailed</i> | Set block size failed. |
| <i>kStatus_SDSPI_SendCsd-Failed</i>      | Send CSD failed.       |
| <i>kStatus_SDSPI_SendCid-Failed</i>      | Send CID failed.       |
| <i>kStatus_Success</i>                   | Operate successfully.  |

#### 45.5.2 void SDSPI\_Deinit ( sdspi\_card\_t \* *card* )

This function deinitializes the specific card.

Parameters

|             |                 |
|-------------|-----------------|
| <i>card</i> | Card descriptor |
|-------------|-----------------|

#### 45.5.3 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. |

#### 45.5.4 status\_t SDSPI\_ReadBlocks ( sdspi\_card\_t \* *card*, uint8\_t \* *buffer*, uint32\_t *startBlock*, uint32\_t *blockCount* )

This function reads blocks from specific card.

## Function Documentation

### 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.     |

### 45.5.5 **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_Write-Protected</i>    | Card is write protected. |
| <i>kStatus_SDSPI_Send-CommandFailed</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.                |

#### 45.5.6 status\_t SDSPI\_SendCid ( sdspi\_card\_t \* *card* )

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

Return values

|                                        |                          |
|----------------------------------------|--------------------------|
| <i>kStatus_SDSPI_SendCommandFailed</i> | Send command failed.     |
| <i>kStatus_SDSPI_ReadFailed</i>        | Read data blocks failed. |
| <i>kStatus_Success</i>                 | Operate successfully.    |

#### 45.5.7 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. |

## Function Documentation

### Return values

|                                                               |                       |
|---------------------------------------------------------------|-----------------------|
| <i>kStatus_SDSPI_Send-<br/>CommandFailed</i>                  | Send command failed.  |
| <i>kStatus_SDSPI_Send-<br/>ApplicationCommand-<br/>Failed</i> |                       |
| <i>kStatus_SDSPI_-<br/>ResponseError</i>                      |                       |
| <i>kStatus_Success</i>                                        | Operate successfully. |

### 45.5.8 **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_Wait-<br/>ReadyFailed</i>   | Wait ready failed.    |
| <i>kStatus_SDSPI_Send-<br/>CommandFailed</i> | Send command failed.  |
| <i>kStatus_Success</i>                       | Operate successfully. |

### 45.5.9 **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 46

### CODEC codec Driver

#### Overview

The MCUXpresso SDK provides a codec abstraction driver interface to access codec register.

#### Modules

- [codec common Driver](#)
- [cs42888 Driver](#)
- [da7212 Driver](#)
- [sgtl5000 Driver](#)
- [wm8904 Driver](#)
- [wm8960 Driver](#)

## codec common Driver

### 46.2.1 Overview

The codec common driver provide codec control abstraction interface.

### 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 `_codec_status` {  
    `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 `_codec_audio_sample_rate` {

```

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 _codec_audio_bit_width {
 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_ModuleMxier = 12U }

 audio codec module
• enum codec_module_ctrl_cmd_t { kCODEC_ModuleSwitchI2SInInterface = 0U }

 audio codec module control cmd
• enum _codec_module_ctrl_i2s_in_interface {
 kCODEC_ModuleI2SInInterfacePCM = 0U,
 kCODEC_ModuleI2SInInterfaceDSD = 1U }

 audio codec module digital interface
• enum _codec_record_source {
 kCODEC_RecordSourceDifferentialLine = 1U,
 kCODEC_RecordSourceLineInput = 2U,
 kCODEC_RecordSourceDifferentialMic = 4U,
 kCODEC_RecordSourceDigitalMic = 8U,
 kCODEC_RecordSourceSingleEndMic = 16U }

```

## codec common Driver

- audio codec module record source value*
  - enum `_codec_reocrd_channel` {
    - `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 `_codec_play_source` {
    - `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 `_codec_play_channel` {
    - `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 `_codec_capability_flag` {

```

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-initialization.*
- [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 recordRource)  
*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, 2, 0))  
*CLOCK driver version 2.2.0.*

## 46.2.2 Data Structure Documentation

### 46.2.2.1 struct codec\_config\_t

#### Data Fields

- uint32\_t [codecDevType](#)  
*codec type*
- void \* [codecDevConfig](#)  
*Codec device specific configuration.*



### 46.2.2.2 struct codec\_capability\_t

#### Data Fields

- uint32\_t [codecModuleCapability](#)  
*codec module capability*
- uint32\_t [codecPlayCapability](#)  
*codec play capability*
- uint32\_t [codecRecordCapability](#)  
*codec record capability*

### 46.2.2.3 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*

## 46.2.3 Macro Definition Documentation

### 46.2.3.1 #define FSL\_CODEC\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 0))

## 46.2.4 Enumeration Type Documentation

### 46.2.4.1 enum \_codec\_status

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.

#### 46.2.4.2 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.

#### 46.2.4.3 enum \_codec\_audio\_sample\_rate

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.

#### 46.2.4.4 enum \_codec\_audio\_bit\_width

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

#### 46.2.4.5 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\_ModuleMxier*** codec module mixer

#### 46.2.4.6 enum codec\_module\_ctrl\_cmd\_t

Enumerator

***kCODEC\_ModuleSwitchI2SInInterface*** module digital interface siwtch.

#### 46.2.4.7 enum \_codec\_module\_ctrl\_i2s\_in\_interface

Enumerator

***kCODEC\_ModuleI2SInInterfacePCM*** Pcm interface.  
***kCODEC\_ModuleI2SInInterfaceDSD*** DSD interface.

#### 46.2.4.8 enum \_codec\_record\_source

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

#### 46.2.4.9 enum \_codec\_reocrd\_channel

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

## codec common Driver

***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

### 46.2.4.10 enum \_codec\_play\_source

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

### 46.2.4.11 enum \_codec\_play\_channel

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

#### 46.2.4.12 enum\_codec\_capability\_flag

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

## codec common Driver

***kCODEC\_SupportRecordChannelRight2*** right record channel 2

***kCODEC\_SupportRecordChannelRight3*** right record channel 3

### 46.2.5 Function Documentation

#### 46.2.5.1 **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.

#### 46.2.5.2 **status\_t CODEC\_Deinit ( codec\_handle\_t \* *handle* )**

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

#### 46.2.5.3 **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.

#### 46.2.5.4 **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.

#### 46.2.5.5 **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 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.

#### 46.2.5.6 **status\_t CODEC\_SetMute ( codec\_handle\_t \* *handle*, uint32\_t *channel*, bool *mute* )**

## codec common Driver

### Parameters

|                |                                                                                                 |
|----------------|-------------------------------------------------------------------------------------------------|
| <i>handle</i>  | codec handle.                                                                                   |
| <i>channel</i> | audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> . |
| <i>mute</i>    | true is mute, false is unmute.                                                                  |

### Returns

kStatus\_Success is success, else configure failed.

#### 46.2.5.7 **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.

#### 46.2.5.8 **status\_t CODEC\_SetRecord ( codec\_handle\_t \* *handle*, uint32\_t *recordRource* )**

### Parameters

|                     |                                                                                                   |
|---------------------|---------------------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                                     |
| <i>recordRource</i> | audio codec record source, can be a value or combine value of <code>_codec_record_source</code> . |

### Returns

kStatus\_Success is success, else configure failed.

#### 46.2.5.9 **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.

#### 46.2.5.10 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.

## cs42888 Driver

### 46.3.1 Overview

The cs42888 driver provide codec control interface.

## Modules

- [cs42888 adapter](#)

## 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](#) 0x01  
*Define the register address of CS42888.*
- #define [CS42888\\_CACHEREGNUM](#) 28  
*Cache register number.*
- #define [CS42888\\_I2C\\_ADDR](#) 0x48  
*CS42888 I2C address.*
- #define [CS42888\\_I2C\\_BITRATE](#) (1000000U)  
*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 `_cs42888_play_channel` {  
`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` bit-Width)  
*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` adc-Mode, `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)

- Get the volume of different AOUT channel in CS42888.*
- `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, 1))`  
*cs42888 driver version 2.1.1.*

## 46.3.2 Data Structure Documentation

### 46.3.2.1 struct cs42888\_audio\_format\_t

#### Data Fields

- `uint32_t mclk_HZ`  
*master clock frequency*
- `uint32_t sampleRate`  
*sample rate*
- `uint32_t bitWidth`  
*bit width*

### 46.3.2.2 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*

#### 46.3.2.2.0.4 Field Documentation

46.3.2.2.0.4.1 `cs42888_func_mode` `cs42888_config_t::ADCMode`

46.3.2.2.0.4.2 `cs42888_func_mode` `cs42888_config_t::DACMode`

#### 46.3.2.3 struct `cs42888_handle_t`

##### Data Fields

- `cs42888_config_t` \* `config`  
*cs42888 config pointer*
- `uint8_t` `i2cHandle` [`CS42888_I2C_HANDLER_SIZE`]  
*i2c handle pointer*

#### 46.3.3 Macro Definition Documentation

46.3.3.1 `#define` `FSL_CS42888_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 1)`)

46.3.3.2 `#define` `CS42888_ID` `0x01`

46.3.3.3 `#define` `CS42888_I2C_ADDR` `0x48`

#### 46.3.4 Enumeration Type Documentation

##### 46.3.4.1 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

### 46.3.4.2 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.

### 46.3.4.3 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.

### 46.3.4.4 enum \_cs42888\_play\_channel

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

## 46.3.5 Function Documentation

### 46.3.5.1 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. |

**46.3.5.2 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. |
|---------------|-----------------------------------|

**46.3.5.3 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                     |

**46.3.5.4 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>   | different working mode of CS42888. |

**46.3.5.5 status\_t CS42888\_SelectFunctionalMode ( cs42888\_handle\_t \* *handle*, cs42888\_func\_mode *adcMode*, cs42888\_func\_mode *dacMode* )**

## cs42888 Driver

### Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>handle</i>  | CS42888 handle structure.          |
| <i>adcMode</i> | different working mode of CS42888. |
| <i>dacMode</i> | different working mode of CS42888. |

#### 46.3.5.6 **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.   |

#### 46.3.5.7 **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.  |

#### 46.3.5.8 **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. |

**46.3.5.9 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. |

**46.3.5.10 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. |

**46.3.5.11 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, false is unmute.   |

**46.3.5.12 status\_t CS42888\_SetModule ( cs42888\_handle\_t \* *handle*, cs42888\_module\_t *module*, bool *isEnabled* )**

## cs42888 Driver

### Parameters

|                  |                            |
|------------------|----------------------------|
| <i>handle</i>    | CS42888 handle structure.  |
| <i>module</i>    | Module expected to enable. |
| <i>isEnabled</i> | Enable or disable moudles. |

#### 46.3.5.13 **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).                                                       |

#### 46.3.5.14 **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. |

#### 46.3.5.15 **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.                |

#### 46.3.5.16 **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.                                          |

### 46.3.6 cs42888 adapter

#### 46.3.6.1 Overview

The cs42888 adapter provide codec unify control interface .

#### Macros

- #define [HAL\\_CODEC\\_HANDLER\\_SIZE](#) ([CS42888\\_I2C\\_HANDLER\\_SIZE](#) + 4)  
*codec handler size*

#### Enumerations

- enum [\\_codec\\_type](#) {  
    [kCODEC\\_CS42888](#),  
    [kCODEC\\_DA7212](#),  
    [kCODEC\\_WM8904](#),  
    [kCODEC\\_WM8960](#),  
    [kCODEC\\_WM8524](#),  
    [kCODEC\\_SGTL5000](#),  
    [kCODEC\\_DA7212](#),  
    [kCODEC\\_CS42888](#),  
    [kCODEC\\_AK4497](#),  
    [kCODEC\\_AK4458](#),  
    [kCODEC\\_TFA9XXX](#),  
    [kCODEC\\_TFA9896](#),  
    [kCODEC\\_SGTL5000](#),  
    [kCODEC\\_WM8904](#),  
    [kCODEC\\_WM8960](#) }  
*codec type*

#### Functions

- [status\\_t HAL\\_CODEC\\_Init](#) (void \*handle, void \*config)  
*Codec initialization.*
- [status\\_t HAL\\_CODEC\\_Deinit](#) (void \*handle)  
*Codec de-initialization.*
- [status\\_t HAL\\_CODEC\\_SetFormat](#) (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bit-Width)  
*set audio data format.*
- [status\\_t HAL\\_CODEC\\_SetVolume](#) (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- [status\\_t HAL\\_CODEC\\_SetMute](#) (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- [status\\_t HAL\\_CODEC\\_SetPower](#) (void \*handle, uint32\_t module, bool powerOn)

- *set audio codec module power.*  
**status\_t HAL\_CODEC\_SetRecord** (void \*handle, uint32\_t recordSource)
- *codec set record source.*  
**status\_t HAL\_CODEC\_SetRecordChannel** (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)
- *codec set record channel.*  
**status\_t HAL\_CODEC\_SetPlay** (void \*handle, uint32\_t playSource)
- *codec set play source.*  
**status\_t HAL\_CODEC\_ModuleControl** (void \*handle, uint32\_t cmd, uint32\_t data)
- *codec module control.*

## 46.3.6.2 Enumeration Type Documentation

### 46.3.6.2.1 enum\_codec\_type

Enumerator

**kCODEC\_CS42888** CS42888.  
**kCODEC\_DA7212** da7212  
**kCODEC\_WM8904** wm8904  
**kCODEC\_WM8960** wm8960  
**kCODEC\_WM8524** wm8524  
**kCODEC\_SGTL5000** sgtl5000  
**kCODEC\_DA7212** da7212  
**kCODEC\_CS42888** CS42888.  
**kCODEC\_AK4497** AK4497.  
**kCODEC\_AK4458** ak4458  
**kCODEC\_TFA9XXX** tfa9xxx  
**kCODEC\_TFA9896** tfa9896  
**kCODEC\_SGTL5000** sgtl5000  
**kCODEC\_WM8904** wm8904  
**kCODEC\_WM8960** wm8960

## 46.3.6.3 Function Documentation

### 46.3.6.3.1 status\_t HAL\_CODEC\_Init ( void \* handle, void \* config )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

## cs42888 Driver

|               |                      |
|---------------|----------------------|
| <i>config</i> | codec configuration. |
|---------------|----------------------|

Returns

kStatus\_Success is success, else initial failed.

### 46.3.6.3.2 status\_t HAL\_CODEC\_Deinit ( void \* *handle* )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

### 46.3.6.3.3 status\_t HAL\_CODEC\_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.

### 46.3.6.3.4 status\_t HAL\_CODEC\_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 <code>_codec_play_channel</code> . |
| <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.

#### 46.3.6.3.5 status\_t HAL\_CODEC\_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 <code>_codec_play_channel</code> . |
| <i>isMute</i>      | true is mute, false is unmute.                                                                  |

Returns

kStatus\_Success is success, else configure failed.

#### 46.3.6.3.6 status\_t HAL\_CODEC\_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.

#### 46.3.6.3.7 status\_t HAL\_CODEC\_SetRecord ( void \* *handle*, uint32\_t *recordSource* )

## cs42888 Driver

### 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.

#### 46.3.6.3.8 status\_t HAL\_CODEC\_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.

#### 46.3.6.3.9 status\_t HAL\_CODEC\_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.

#### 46.3.6.3.10 status\_t HAL\_CODEC\_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 <code>_codec_module_ctrl_cmd</code> .                                                                                                                                                                                                                               |
| <i>data</i>   | value to write, when cmd is <code>kCODEC_ModuleRecordSourceChannel</code> , the data should be a value combine of channel and source, please reference macro <code>CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN)</code> , reference codec specific driver for detail configurations. |

## Returns

`kStatus_Success` is success, else configure failed.

## da7212 Driver

## da7212 Driver

### 46.4.1 Overview

The da7212 driver provide codec control interface.

## Modules

- [da7212 adapter](#)

## 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.*

## 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` {

## da7212 Driver

```
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 *config)`  
*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 fl 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 module, uint32_t volume)`  
*Set module volume.*
- `status_t DA7212_SetChannelMute (da7212_handle_t *handle, uint32_t module, 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, 0))`  
*CLOCK driver version 2.2.0.*

## 46.4.2 Data Structure Documentation

### 46.4.2.1 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*

### 46.4.2.2 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 interval*

### 46.4.2.3 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*

## da7212 Driver

### 46.4.2.3.0.1 Field Documentation

46.4.2.3.0.1.1 `bool da7212_config_t::isMaster`

46.4.2.3.0.1.2 `da7212_protocol_t da7212_config_t::protocol`

46.4.2.3.0.1.3 `da7212_dac_source_t da7212_config_t::dacSource`

### 46.4.2.4 `struct da7212_handle_t`

#### Data Fields

- `da7212_config_t * config`  
*da7212 config pointer*
- `uint8_t i2cHandle [DA7212_I2C_HANDLER_SIZE]`  
*i2c handle*

### 46.4.3 Macro Definition Documentation

46.4.3.1 `#define FSL_DA7212_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))`

### 46.4.4 Enumeration Type Documentation

#### 46.4.4.1 `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.

#### 46.4.4.2 `enum _da7212_play_channel`

Enumerator

*kDA7212\_HeadphoneLeft* headphone left  
*kDA7212\_HeadphoneRight* headphone right  
*kDA7212\_Speaker* speaker channel

#### 46.4.4.3 `enum da7212_Output_t`

Enumerator

*kDA7212\_Output\_HP* Output to headphone.

***kDA7212\_Output\_SP*** Output to speaker.

#### 46.4.4.4 enum \_da7212\_module

Enumerator

***kDA7212\_ModuleADC*** module ADC  
***kDA7212\_ModuleDAC*** module DAC  
***kDA7212\_ModuleHeadphone*** module headphone  
***kDA7212\_ModuleSpeaker*** module speaker

#### 46.4.4.5 enum da7212\_dac\_source\_t

Enumerator

***kDA7212\_DACSourceADC*** DAC source from ADC.  
***kDA7212\_DACSourceInputStream*** DAC source from.

#### 46.4.4.6 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.

#### 46.4.4.7 enum da7212\_protocol\_t

Enumerator

***kDA7212\_BusI2S*** I2S Type.

## da7212 Driver

*kDA7212\_BusLeftJustified* Left justified.  
*kDA7212\_BusRightJustified* Right Justified.  
*kDA7212\_BusDSPMode* DSP mode.

### 46.4.4.8 enum da7212\_sys\_clk\_source\_t

Enumerator

*kDA7212\_SysClkSourceMCLK* da7212 system clock source from MCLK  
*kDA7212\_SysClkSourcePLL* da7212 system clock source from PLL

### 46.4.4.9 enum da7212\_pll\_clk\_source\_t

Enumerator

*kDA7212\_PLLClkSourceMCLK* DA7212 PLL clock source from MCLK.

### 46.4.4.10 enum da7212\_pll\_out\_clk\_t

Enumerator

*kDA7212\_PLLOutputClk11289600* output 112896000U  
*kDA7212\_PLLOutputClk12288000* output 12288000U

### 46.4.4.11 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

## 46.4.5 Function Documentation

46.4.5.1 status\_t DA7212\_Init ( da7212\_handle\_t \* *handle*, da7212\_config\_t \* *config* )



## Parameters

|               |                                                                                                                                                                                                                                                                                 |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | DA7212 handle pointer.                                                                                                                                                                                                                                                          |
| <i>config</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 = kDA7212_BusI2S * codec_config.isMaster = false * </pre> |

#### 46.4.5.2 **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 while 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.                                                                                                                                                        |

#### 46.4.5.3 **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. |

#### 46.4.5.4 **void DA7212\_ChangeHPVolume ( da7212\_handle\_t \* *handle*, da7212\_volume\_t *volume* )**

## da7212 Driver

### Parameters

|               |                         |
|---------------|-------------------------|
| <i>handle</i> | DA7212 handle pointer.  |
| <i>volume</i> | The volume of playback. |

#### 46.4.5.5 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. |

#### 46.4.5.6 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.     |

#### 46.4.5.7 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. |

#### 46.4.5.8 status\_t DA7212\_SetChannelVolume ( da7212\_handle\_t \* *handle*, uint32\_t *module*, uint32\_t *volume* )

### Parameters

|               |                                                            |
|---------------|------------------------------------------------------------|
| <i>handle</i> | DA7212 handle pointer.                                     |
| <i>module</i> | shoule be a value of _da7212_module                        |
| <i>volume</i> | volume range 0 - 100, 0 is mute, 100 is the maximum value. |

#### 46.4.5.9 **status\_t DA7212\_SetChannelMute ( da7212\_handle\_t \* *handle*, uint32\_t *module*, bool *isMute* )**

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>handle</i> | DA7212 handle pointer.              |
| <i>module</i> | shoule be a value of _da7212_module |
| <i>isMute</i> | true is mute, false is unmute.      |

#### 46.4.5.10 **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.     |

#### 46.4.5.11 **status\_t DA7212\_SetMasterModeBits ( da7212\_handle\_t \* *handle*, uint32\_t *bitWidth* )**

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>handle</i>   | DA7212 handle pointer. |
| <i>bitWidth</i> | audio data bitwidth.   |

#### 46.4.5.12 **status\_t DA7212\_WriteRegister ( da7212\_handle\_t \* *handle*, uint8\_t *u8Register*, uint8\_t *u8RegisterData* )**

## da7212 Driver

### 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       |

**46.4.5.13** `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>pu8Register-Data</i> | Pointer where the read out value to be stored. |

**46.4.5.14** `status_t DA7212_Deinit ( da7212_handle_t * handle )`

### Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | DA7212 handle pointer. |
|---------------|------------------------|

## 46.4.6 da7212 adapter

### 46.4.6.1 Overview

The da7212 adapter provide codec unify control interface .

#### Macros

- #define `HAL_CODEC_HANDLER_SIZE` (`DA7212_I2C_HANDLER_SIZE` + 4)  
*codec handler size*

#### Enumerations

- enum `_codec_type` {  
`kCODEC_CS42888`,  
`kCODEC_DA7212`,  
`kCODEC_WM8904`,  
`kCODEC_WM8960`,  
`kCODEC_WM8524`,  
`kCODEC_SGTL5000`,  
`kCODEC_DA7212`,  
`kCODEC_CS42888`,  
`kCODEC_AK4497`,  
`kCODEC_AK4458`,  
`kCODEC_TFA9XXX`,  
`kCODEC_TFA9896`,  
`kCODEC_SGTL5000`,  
`kCODEC_WM8904`,  
`kCODEC_WM8960` }  
*codec type*

#### Functions

- `status_t HAL_CODEC_Init` (void \*handle, void \*config)  
*Codec initialization.*
- `status_t HAL_CODEC_Deinit` (void \*handle)  
*Codec de-initialization.*
- `status_t HAL_CODEC_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bit-Width)  
*set audio data format.*
- `status_t HAL_CODEC_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- `status_t HAL_CODEC_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- `status_t HAL_CODEC_SetPower` (void \*handle, uint32\_t module, bool powerOn)

## da7212 Driver

- set audio codec module power.*
  - [status\\_t HAL\\_CODEC\\_SetRecord](#) (void \*handle, uint32\_t recordSource)
- codec set record source.*
  - [status\\_t HAL\\_CODEC\\_SetRecordChannel](#) (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)
- codec set record channel.*
  - [status\\_t HAL\\_CODEC\\_SetPlay](#) (void \*handle, uint32\_t playSource)
- codec set play source.*
  - [status\\_t HAL\\_CODEC\\_ModuleControl](#) (void \*handle, uint32\_t cmd, uint32\_t data)
- codec module control.*

### 46.4.6.2 Enumeration Type Documentation

#### 46.4.6.2.1 enum\_codec\_type

Enumerator

**kCODEC\_CS42888** CS42888.  
**kCODEC\_DA7212** da7212  
**kCODEC\_WM8904** wm8904  
**kCODEC\_WM8960** wm8960  
**kCODEC\_WM8524** wm8524  
**kCODEC\_SGTL5000** sgtl5000  
**kCODEC\_DA7212** da7212  
**kCODEC\_CS42888** CS42888.  
**kCODEC\_AK4497** AK4497.  
**kCODEC\_AK4458** ak4458  
**kCODEC\_TFA9XXX** tfa9xxx  
**kCODEC\_TFA9896** tfa9896  
**kCODEC\_SGTL5000** sgtl5000  
**kCODEC\_WM8904** wm8904  
**kCODEC\_WM8960** wm8960

### 46.4.6.3 Function Documentation

#### 46.4.6.3.1 status\_t HAL\_CODEC\_Init ( void \* *handle*, void \* *config* )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

|               |                      |
|---------------|----------------------|
| <i>config</i> | codec configuration. |
|---------------|----------------------|

Returns

kStatus\_Success is success, else initial failed.

#### 46.4.6.3.2 status\_t HAL\_CODEC\_Deinit ( void \* *handle* )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

#### 46.4.6.3.3 status\_t HAL\_CODEC\_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.

#### 46.4.6.3.4 status\_t HAL\_CODEC\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

Parameters

## da7212 Driver

|                    |                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                                   |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> . |
| <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.

#### 46.4.6.3.5 `status_t HAL_CODEC_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 <code>_codec_play_channel</code> . |
| <i>isMute</i>      | true is mute, false is unmute.                                                                  |

### Returns

`kStatus_Success` is success, else configure failed.

#### 46.4.6.3.6 `status_t HAL_CODEC_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.

#### 46.4.6.3.7 `status_t HAL_CODEC_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.

#### 46.4.6.3.8 status\_t HAL\_CODEC\_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.

#### 46.4.6.3.9 status\_t HAL\_CODEC\_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.

#### 46.4.6.3.10 status\_t HAL\_CODEC\_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

## da7212 Driver

### Parameters

|               |                                                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                                                     |
| <i>cmd</i>    | module control cmd, reference <code>_codec_module_ctrl_cmd</code> .                                                                                                                                                                                                                               |
| <i>data</i>   | value to write, when cmd is <code>kCODEC_ModuleRecordSourceChannel</code> , the data should be a value combine of channel and source, please reference macro <code>CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN)</code> , reference codec specific driver for detail configurations. |

### Returns

`kStatus_Success` is success, else configure failed.

## sgtl5000 Driver

### 46.5.1 Overview

The sgtl5000 driver provide codec control interface.

### Modules

- [sgtl5000 adapter](#)

### Data Structures

- struct [sgtl\\_audio\\_format\\_t](#)  
*Audio format configuration. [More...](#)*
- struct [sgtl\\_config\\_t](#)  
*Initailize structure of sgtl5000. [More...](#)*
- struct [sgtl\\_handle\\_t](#)  
*SGTL codec handler. [More...](#)*

### Macros

- #define [CHIP\\_ID](#) 0x0000  
*Define the register address of sgtl5000.*
- #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 Sgtl5000 board.*

## sgtl5000 Driver

- 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 `_sgtl_play_channel` {  
    `kSGTL_HeadphoneLeft` = 0,  
    `kSGTL_HeadphoneRight` = 1,  
    `kSGTL_LineoutLeft` = 2,  
    `kSGTL_LineoutRight` = 3 }  
    *sgtl play channel*
- enum `_sgtl_record_source` {  
    `kSGTL_RecordSourceLineIn` = 0U,  
    `kSGTL_RecordSourceMic` = 1U }  
    *sgtl record source*
- enum `_stgl_play_source` {  
    `kSGTL_PlaySourceLineIn` = 0U,  
    `kSGTL_PlaySourceDAC` = 1U }  
    *sgtl play source*
- enum `sgtl_sclk_edge_t` {  
    `kSGTL_SclkValidEdgeRising` = 0U,  
    `kSGTL_SclkValidEdgeFalling` = 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, 0))`  
*CLOCK driver version 2.1.0.*

## 46.5.2 Data Structure Documentation

### 46.5.2.1 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*

### 46.5.2.2 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*

#### 46.5.2.2.0.1 Field Documentation

##### 46.5.2.2.0.1.1 sgtl\_route\_t sgtl\_config\_t::route

##### 46.5.2.2.0.1.2 bool sgtl\_config\_t::master\_slave

True means master, false means slave.

### 46.5.2.3 struct sgtl\_handle\_t

#### Data Fields

- [sgtl\\_config\\_t](#) \* config  
*sgtl config pointer*
- uint8\_t [i2cHandle](#) [SGTL\_I2C\_HANDLER\_SIZE]  
*i2c handle*

### 46.5.3 Macro Definition Documentation

**46.5.3.1** `#define FSL_SGTL5000_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

**46.5.3.2** `#define CHIP_ID 0x0000`

**46.5.3.3** `#define SGTL5000_I2C_ADDR 0x0A`

### 46.5.4 Enumeration Type Documentation

#### 46.5.4.1 `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 module in SGTL5000.  
*kSGTL\_ModuleLineOut* Line-out module in SGTL5000.  
*kSGTL\_ModuleMicin* Micphone module in SGTL5000.

#### 46.5.4.2 `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.

#### 46.5.4.3 `enum sgtl_protocol_t`

Sgtl5000 only supports I2S format and PCM format.

## sgtl5000 Driver

### Enumerator

*kSGTL\_BusI2S* I2S Type.  
*kSGTL\_BusLeftJustified* Left justified.  
*kSGTL\_BusRightJustified* Right Justified.  
*kSGTL\_BusPCMA* PCMA.  
*kSGTL\_BusPCMB* PCMB.

#### 46.5.4.4 enum \_sgtl\_play\_channel

### Enumerator

*kSGTL\_HeadphoneLeft* headphone left channel  
*kSGTL\_HeadphoneRight* headphone right channel  
*kSGTL\_LineoutLeft* lineout left channel  
*kSGTL\_LineoutRight* lineout right channel

#### 46.5.4.5 enum \_sgtl\_record\_source

### Enumerator

*kSGTL\_RecordSourceLineIn* record source line in  
*kSGTL\_RecordSourceMic* record source single end

#### 46.5.4.6 enum \_stgl\_play\_source

### Enumerator

*kSGTL\_PlaySourceLineIn* play source line in  
*kSGTL\_PlaySourceDAC* play source line in

#### 46.5.4.7 enum sgtl\_sclk\_edge\_t

### Enumerator

*kSGTL\_SclkValidEdgeRising* SCLK valid edge.  
*kSGTL\_SclkValidEdgeFailling* SCLK failling edge.

### 46.5.5 Function Documentation

#### 46.5.5.1 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

#### 46.5.5.2 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. |

#### 46.5.5.3 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.

## sgtl5000 Driver

### Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>handle</i>   | Sgtl5000 handle structure.    |
| <i>protocol</i> | Audio data transfer protocol. |

### 46.5.5.4 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. |

### 46.5.5.5 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.

#### 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. |

### 46.5.5.6 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.

**46.5.5.7** `status_t SGTL_SetMute ( sgtl_handle_t * handle, sgtl_module_t module, bool mute )`

## sgtl5000 Driver

### 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.   |

### 46.5.5.8 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. |

### 46.5.5.9 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. |

### 46.5.5.10 status\_t SGTL\_Deinit ( sgtl\_handle\_t \* *handle* )

Shut down Sctl5000 modules.

#### Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>handle</i> | Sgtl5000 handle structure pointer. |
|---------------|------------------------------------|

### 46.5.5.11 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).                                                        |

#### 46.5.5.12 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.

#### 46.5.5.13 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.

#### 46.5.5.14 status\_t SGTL\_WriteReg ( sgtl\_handle\_t \* *handle*, uint16\_t *reg*, uint16\_t *val* )

Parameters

|               |                            |
|---------------|----------------------------|
| <i>handle</i> | Sgtl5000 handle structure. |
|---------------|----------------------------|

## sgtl5000 Driver

|            |                                         |
|------------|-----------------------------------------|
| <i>reg</i> | The register address in sgtl.           |
| <i>val</i> | Value needs to write into the register. |

### 46.5.5.15 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.             |

### 46.5.5.16 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.                                          |

## 46.5.6 sgtl5000 adapter

### 46.5.6.1 Overview

The sgtl5000 adapter provide codec unify control interface .

#### Macros

- #define `HAL_CODEC_HANDLER_SIZE` (`SGTL_I2C_HANDLER_SIZE` + 4)  
*codec handler size*

#### Enumerations

- enum `_codec_type` {  
`kCODEC_CS42888`,  
`kCODEC_DA7212`,  
`kCODEC_WM8904`,  
`kCODEC_WM8960`,  
`kCODEC_WM8524`,  
`kCODEC_SGTL5000`,  
`kCODEC_DA7212`,  
`kCODEC_CS42888`,  
`kCODEC_AK4497`,  
`kCODEC_AK4458`,  
`kCODEC_TFA9XXX`,  
`kCODEC_TFA9896`,  
`kCODEC_SGTL5000`,  
`kCODEC_WM8904`,  
`kCODEC_WM8960` }  
*codec type*

#### Functions

- `status_t HAL_CODEC_Init` (void \*handle, void \*config)  
*Codec initialization.*
- `status_t HAL_CODEC_Deinit` (void \*handle)  
*Codec de-initialization.*
- `status_t HAL_CODEC_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bit-Width)  
*set audio data format.*
- `status_t HAL_CODEC_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- `status_t HAL_CODEC_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- `status_t HAL_CODEC_SetPower` (void \*handle, uint32\_t module, bool powerOn)

## sgtl5000 Driver

- set audio codec module power.*
  - [status\\_t HAL\\_CODEC\\_SetRecord](#) (void \*handle, uint32\_t recordSource)  
*codec set record source.*
  - [status\\_t HAL\\_CODEC\\_SetRecordChannel](#) (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
  - [status\\_t HAL\\_CODEC\\_SetPlay](#) (void \*handle, uint32\_t playSource)  
*codec set play source.*
  - [status\\_t HAL\\_CODEC\\_ModuleControl](#) (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

### 46.5.6.2 Enumeration Type Documentation

#### 46.5.6.2.1 enum\_codec\_type

Enumerator

**kCODEC\_CS42888** CS42888.  
**kCODEC\_DA7212** da7212  
**kCODEC\_WM8904** wm8904  
**kCODEC\_WM8960** wm8960  
**kCODEC\_WM8524** wm8524  
**kCODEC\_SGTL5000** sgtl5000  
**kCODEC\_DA7212** da7212  
**kCODEC\_CS42888** CS42888.  
**kCODEC\_AK4497** AK4497.  
**kCODEC\_AK4458** ak4458  
**kCODEC\_TFA9XXX** tfa9xxx  
**kCODEC\_TFA9896** tfa9896  
**kCODEC\_SGTL5000** sgtl5000  
**kCODEC\_WM8904** wm8904  
**kCODEC\_WM8960** wm8960

### 46.5.6.3 Function Documentation

#### 46.5.6.3.1 status\_t HAL\_CODEC\_Init ( void \* *handle*, void \* *config* )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|



|               |                      |
|---------------|----------------------|
| <i>config</i> | codec configuration. |
|---------------|----------------------|

Returns

kStatus\_Success is success, else initial failed.

#### 46.5.6.3.2 status\_t HAL\_CODEC\_Deinit ( void \* *handle* )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

#### 46.5.6.3.3 status\_t HAL\_CODEC\_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.

#### 46.5.6.3.4 status\_t HAL\_CODEC\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

Parameters

## sgtl5000 Driver

|                    |                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                                   |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> . |
| <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.

#### 46.5.6.3.5 status\_t HAL\_CODEC\_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 <code>_codec_play_channel</code> . |
| <i>isMute</i>      | true is mute, false is unmute.                                                                  |

### Returns

kStatus\_Success is success, else configure failed.

#### 46.5.6.3.6 status\_t HAL\_CODEC\_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.

#### 46.5.6.3.7 status\_t HAL\_CODEC\_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.

#### 46.5.6.3.8 status\_t HAL\_CODEC\_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.

#### 46.5.6.3.9 status\_t HAL\_CODEC\_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.

#### 46.5.6.3.10 status\_t HAL\_CODEC\_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

## sgtl5000 Driver

### Parameters

|               |                                                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                                                     |
| <i>cmd</i>    | module control cmd, reference <code>_codec_module_ctrl_cmd</code> .                                                                                                                                                                                                                               |
| <i>data</i>   | value to write, when cmd is <code>kCODEC_ModuleRecordSourceChannel</code> , the data should be a value combine of channel and source, please reference macro <code>CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN)</code> , reference codec specific driver for detail configurations. |

### Returns

`kStatus_Success` is success, else configure failed.

## wm8960 Driver

### 46.6.1 Overview

The wm8960 driver provide codec control interface.

### Modules

- [wm8960 adapter](#)

### Data Structures

- struct [wm8960\\_audio\\_format\\_t](#)  
*wm8960 audio format [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](#) 0x0  
*Define the register address of WM8960.*
- #define [WM8960\\_CACHEREGNUM](#) 56  
*Cache register number.*
- #define [WM8960\\_IFACE1\\_FORMAT\\_MASK](#) 0x03  
*WM8960\_IFACE1 FORMAT bits.*
- #define [WM8960\\_IFACE1\\_WL\\_MASK](#) 0x0C  
*WM8960\_IFACE1 WL bits.*
- #define [WM8960\\_IFACE1\\_LRP\\_MASK](#) 0x10  
*WM8960\_IFACE1 LRP bit.*
- #define [WM8960\\_IFACE1\\_DLRSWAP\\_MASK](#) 0x20  
*WM8960\_IFACE1 DLRSWAP bit.*
- #define [WM8960\\_IFACE1\\_MS\\_MASK](#) 0x40  
*WM8960\_IFACE1 MS bit.*
- #define [WM8960\\_IFACE1\\_BCLKINV\\_MASK](#) 0x80  
*WM8960\_IFACE1 BCLKINV bit.*
- #define [WM8960\\_IFACE1\\_ALRSWAP\\_MASK](#) 0x100  
*WM8960\_IFACE1 ALRSWAP bit.*
- #define [WM8960\\_POWER1\\_VREF\\_MASK](#) 0x40  
*WM8960\_POWER1.*
- #define [WM8960\\_POWER2\\_DACL\\_MASK](#) 0x100  
*WM8960\_POWER2.*
- #define [WM8960\\_I2C\\_ADDR](#) 0x1A  
*WM8960 I2C address.*

## wm8960 Driver

- #define WM8960\_I2C\_BAUDRATE (100000U)  
*WM8960 I2C baudrate.*

## 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 \_wm8960\_play\_channel {  
    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 _wm8960_sample_rate {
 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 _wm8960_audio_bit_width {
 kWM8960_AudioBitWidth16bit = 16U,
 kWM8960_AudioBitWidth20bit = 20U,
 kWM8960_AudioBitWidth24bit = 24U,
 kWM8960_AudioBitWidth32bit = 32U }
 audio bit width

```

## Functions

- `status_t WM8960_Init (wm8960_handle_t *handle, const wm8960_config_t *wm8960Config)`  
*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)`

## wm8960 Driver

- *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, 1, 1))  
*CLOCK driver version 2.1.1.*

## 46.6.2 Data Structure Documentation

### 46.6.2.1 struct wm8960\_audio\_format\_t

#### Data Fields

- uint32\_t **mclk\_HZ**  
*master clock frequency*
- uint32\_t **sampleRate**  
*sample rate*
- uint32\_t **bitWidth**  
*bit width*



### 46.6.2.2 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.*
- 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*

#### 46.6.2.2.0.1 Field Documentation

##### 46.6.2.2.0.1.1 [wm8960\\_route\\_t wm8960\\_config\\_t::route](#)

##### 46.6.2.2.0.1.2 [bool wm8960\\_config\\_t::master\\_slave](#)

### 46.6.2.3 struct wm8960\_handle\_t

#### Data Fields

- const [wm8960\\_config\\_t \\* config](#)  
*wm8904 config pointer*
- uint8\_t [i2cHandle](#) [[WM8960\\_I2C\\_HANDLER\\_SIZE](#)]  
*i2c handle*

### 46.6.3 Macro Definition Documentation

46.6.3.1 **#define WM8960\_LINVOL 0x0**

46.6.3.2 **#define WM8960\_I2C\_ADDR 0x1A**

### 46.6.4 Enumeration Type Documentation

#### 46.6.4.1 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.

#### 46.6.4.2 enum \_wm8960\_play\_channel

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

#### 46.6.4.3 enum wm8960\_play\_source\_t

Enumerator

*kWM8960\_PlaySourcePGA* wm8960 play source PGA

*kWM8960\_PlaySourceInput* wm8960 play source Input

*kWM8960\_PlaySourceDAC* wm8960 play source DAC

#### 46.6.4.4 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.

#### 46.6.4.5 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.

#### 46.6.4.6 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.

#### 46.6.4.7 enum \_wm8960\_sample\_rate

Enumerator

*kWM8960\_AudioSampleRate8KHz* Sample rate 8000 Hz.  
*kWM8960\_AudioSampleRate11025Hz* Sample rate 11025 Hz.  
*kWM8960\_AudioSampleRate12KHz* Sample rate 12000 Hz.

## wm8960 Driver

*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.

### 46.6.4.8 enum \_wm8960\_audio\_bit\_width

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

## 46.6.5 Function Documentation

### 46.6.5.1 status\_t WM8960\_Init ( wm8960\_handle\_t \* *handle*, const wm8960\_config\_t \* *wm8960Config* )

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>wm8960Config</i> | WM8960 configuration structure. |

### 46.6.5.2 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. |
|---------------|----------------------------------|

#### 46.6.5.3 **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. |

#### 46.6.5.4 **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.      |

#### 46.6.5.5 **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.      |

#### 46.6.5.6 **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.

## wm8960 Driver

### Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>handle</i>   | WM8960 handle structure.      |
| <i>protocol</i> | Audio data transfer protocol. |

#### 46.6.5.7 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. |

#### 46.6.5.8 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.

### 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.                                   |

#### 46.6.5.9 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.

**46.6.5.10** `status_t WM8960_SetMute ( wm8960_handle_t * handle, wm8960_module_t module, bool isEnabled )`

## wm8960 Driver

### Parameters

|                  |                                   |
|------------------|-----------------------------------|
| <i>handle</i>    | WM8960 handle structure.          |
| <i>module</i>    | Modules need to be mute.          |
| <i>isEnabled</i> | Mute or unmute, 1 represent mute. |

#### 46.6.5.11 **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. |

#### 46.6.5.12 **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_ModuleHeadphoneSourcePGA, kWM8960_ModuleHeadphoneSourceDAC, kWM8960_ModulePlaySourceInput, kWM8960_ModulePlayMonoRight, kWM8960_ModulePlayMonoLeft. |

### Returns

kStatus\_WM8904\_Success if successful, different code otherwise..

#### 46.6.5.13 **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).                                                      |

**46.6.5.14** `status_t WM8960_SetJackDetect ( wm8960_handle_t * handle, bool isEnabled )`

Parameters

|                  |                            |
|------------------|----------------------------|
| <i>handle</i>    | WM8960 handle structure.   |
| <i>isEnabled</i> | Enable or disable moudles. |

**46.6.5.15** `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. |

**46.6.5.16** `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.               |

**46.6.5.17** `status_t WM8960_ModifyReg ( wm8960_handle_t * handle, uint8_t reg, uint16_t mask, uint16_t val )`

## wm8960 Driver

### 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.                                          |

## 46.6.6 wm8960 adapter

### 46.6.6.1 Overview

The wm8960 adapter provide codec unify control interface .

#### Macros

- #define `HAL_CODEC_HANDLER_SIZE` (`WM8960_I2C_HANDLER_SIZE` + 4)  
*codec handler size*

#### Enumerations

- enum `_codec_type` {  
`kCODEC_CS42888`,  
`kCODEC_DA7212`,  
`kCODEC_WM8904`,  
`kCODEC_WM8960`,  
`kCODEC_WM8524`,  
`kCODEC_SGTL5000`,  
`kCODEC_DA7212`,  
`kCODEC_CS42888`,  
`kCODEC_AK4497`,  
`kCODEC_AK4458`,  
`kCODEC_TFA9XXX`,  
`kCODEC_TFA9896`,  
`kCODEC_SGTL5000`,  
`kCODEC_WM8904`,  
`kCODEC_WM8960` }  
*codec type*

#### Functions

- `status_t HAL_CODEC_Init` (void \*handle, void \*config)  
*Codec initialization.*
- `status_t HAL_CODEC_Deinit` (void \*handle)  
*Codec de-initialization.*
- `status_t HAL_CODEC_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bit-Width)  
*set audio data format.*
- `status_t HAL_CODEC_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- `status_t HAL_CODEC_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- `status_t HAL_CODEC_SetPower` (void \*handle, uint32\_t module, bool powerOn)

## wm8960 Driver

- set audio codec module power.*
  - [status\\_t HAL\\_CODEC\\_SetRecord](#) (void \*handle, uint32\_t recordSource)
- codec set record source.*
  - [status\\_t HAL\\_CODEC\\_SetRecordChannel](#) (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)
- codec set record channel.*
  - [status\\_t HAL\\_CODEC\\_SetPlay](#) (void \*handle, uint32\_t playSource)
- codec set play source.*
  - [status\\_t HAL\\_CODEC\\_ModuleControl](#) (void \*handle, uint32\_t cmd, uint32\_t data)
- codec module control.*

### 46.6.6.2 Enumeration Type Documentation

#### 46.6.6.2.1 enum\_codec\_type

Enumerator

**kCODEC\_CS42888** CS42888.  
**kCODEC\_DA7212** da7212  
**kCODEC\_WM8904** wm8904  
**kCODEC\_WM8960** wm8960  
**kCODEC\_WM8524** wm8524  
**kCODEC\_SGTL5000** sgtl5000  
**kCODEC\_DA7212** da7212  
**kCODEC\_CS42888** CS42888.  
**kCODEC\_AK4497** AK4497.  
**kCODEC\_AK4458** ak4458  
**kCODEC\_TFA9XXX** tfa9xxx  
**kCODEC\_TFA9896** tfa9896  
**kCODEC\_SGTL5000** sgtl5000  
**kCODEC\_WM8904** wm8904  
**kCODEC\_WM8960** wm8960

### 46.6.6.3 Function Documentation

#### 46.6.6.3.1 status\_t HAL\_CODEC\_Init ( void \* *handle*, void \* *config* )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

|               |                      |
|---------------|----------------------|
| <i>config</i> | codec configuration. |
|---------------|----------------------|

Returns

kStatus\_Success is success, else initial failed.

#### 46.6.6.3.2 status\_t HAL\_CODEC\_Deinit ( void \* *handle* )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

#### 46.6.6.3.3 status\_t HAL\_CODEC\_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.

#### 46.6.6.3.4 status\_t HAL\_CODEC\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

Parameters

## wm8960 Driver

|                    |                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                                   |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> . |
| <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.

#### 46.6.6.3.5 `status_t HAL_CODEC_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 <code>_codec_play_channel</code> . |
| <i>isMute</i>      | true is mute, false is unmute.                                                                  |

### Returns

`kStatus_Success` is success, else configure failed.

#### 46.6.6.3.6 `status_t HAL_CODEC_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.

#### 46.6.6.3.7 `status_t HAL_CODEC_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.

#### 46.6.6.3.8 status\_t HAL\_CODEC\_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.

#### 46.6.6.3.9 status\_t HAL\_CODEC\_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.

#### 46.6.6.3.10 status\_t HAL\_CODEC\_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

## wm8960 Driver

### Parameters

|               |                                                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                                                     |
| <i>cmd</i>    | module control cmd, reference <code>_codec_module_ctrl_cmd</code> .                                                                                                                                                                                                                               |
| <i>data</i>   | value to write, when cmd is <code>kCODEC_ModuleRecordSourceChannel</code> , the data should be a value combine of channel and source, please reference macro <code>CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN)</code> , reference codec specific driver for detail configurations. |

### Returns

`kStatus_Success` is success, else configure failed.



## wm8904 Driver

### 46.7.1 Overview

The wm8904 driver provide codec control interface.

### Modules

- [wm8904 adapter](#)

### Data Structures

- struct [wm8904\\_fl\\_config\\_t](#)  
*wm8904 fl 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.*

### Enumerations

- enum [\\_wm8904\\_status](#) {  
  [kStatus\\_WM8904\\_Success](#) = 0x0,  
  [kStatus\\_WM8904\\_Fail](#) = 0x1 }  
*WM8904 status return codes.*
- enum [\\_wm8904\\_lrc\\_polarity](#) {  
  [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`  
    *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 }  
    *Sample rate.*
- enum `wm8904_bit_width_t` {  
    `kWM8904_BitWidth16` = 0x0,  
    `kWM8904_BitWidth20` = 0x1,  
    `kWM8904_BitWidth24` = 0x2,  
    `kWM8904_BitWidth32` = 0x3 }  
    *Bit width.*

- enum `_wm8904_record_source` {  
`kWM8904_RecordSourceDifferentialLine` = 1U,  
`kWM8904_RecordSourceLineInput` = 2U,  
`kWM8904_RecordSourceDifferentialMic` = 4U,  
`kWM8904_RecordSourceDigitalMic` = 8U }  
*wm8904 record source*
- enum `_wm8904_record_channel` {  
`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 `_wm8904_play_source` {  
`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_flr_clk_source_t` { `kWM8904_FLLClkSourceMCLK` = 0U }  
*wm8904 flr 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 read 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` \*wm8904\_config)  
*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)

## wm8904 Driver

- Sets WM8904 as master or slave.*
- `status_t WM8904_SeMasterClock` (`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_fl_config_t` \*config)  
*WM8904 set PLL configuration This function will enable the GPIO1 FLL clock output function, so user can see the generated fl 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)  
*brief SET the module output power.*
- `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, 4, 1))  
*WM8904 driver version 2.4.1.*

## 46.7.2 Data Structure Documentation

### 46.7.2.1 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*

### 46.7.2.2 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.*

### 46.7.2.3 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*

## wm8904 Driver

- `uint8_t slaveAddress`  
*code device slave address*
- `codec_i2c_config_t i2cConfig`  
*i2c bus configuration*

### 46.7.2.4 struct wm8904\_handle\_t

#### Data Fields

- `wm8904_config_t * config`  
*wm8904 config pointer*
- `uint8_t i2cHandle [WM8904_I2C_HANDLER_SIZE]`  
*i2c handle*

### 46.7.3 Macro Definition Documentation

46.7.3.1 `#define FSL_WM8904_DRIVER_VERSION (MAKE_VERSION(2, 4, 1))`

46.7.3.2 `#define WM8904_I2C_ADDRESS (0x1A)`

46.7.3.3 `#define WM8904_I2C_BITRATE (400000U)`

### 46.7.4 Enumeration Type Documentation

#### 46.7.4.1 enum \_wm8904\_status

Enumerator

*kStatus\_WM8904\_Success* Success.  
*kStatus\_WM8904\_Fail* Failure.

#### 46.7.4.2 enum \_wm8904\_lrc\_polarity

Enumerator

*kWM8904\_LRCPolarityNormal* LRC polarity normal.  
*kWM8904\_LRCPolarityInverted* LRC polarity inverted.

#### 46.7.4.3 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

#### 46.7.4.4 enum wm8904\_timeslot\_t

Enumerator

*kWM8904\_TimeSlot0* time slot0  
*kWM8904\_TimeSlot1* time slot1

#### 46.7.4.5 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.

#### 46.7.4.6 enum wm8904\_fs\_ratio\_t

Enumerator

*kWM8904\_FsRatio64X* SYSCLK is 64 \* sample rate \* frame width.  
*kWM8904\_FsRatio128X* SYSCLK is 128 \* sample rate \* frame width.  
*kWM8904\_FsRatio192X* SYSCLK is 192 \* sample rate \* frame width.  
*kWM8904\_FsRatio256X* SYSCLK is 256 \* sample rate \* frame width.  
*kWM8904\_FsRatio384X* SYSCLK is 384 \* sample rate \* frame width.  
*kWM8904\_FsRatio512X* SYSCLK is 512 \* sample rate \* frame width.  
*kWM8904\_FsRatio768X* SYSCLK is 768 \* sample rate \* frame width.  
*kWM8904\_FsRatio1024X* SYSCLK is 1024 \* sample rate \* frame width.  
*kWM8904\_FsRatio1408X* SYSCLK is 1408 \* sample rate \* frame width.  
*kWM8904\_FsRatio1536X* SYSCLK is 1536 \* sample rate \* frame width.

#### 46.7.4.7 enum wm8904\_sample\_rate\_t

Enumerator

*kWM8904\_SampleRate8kHz* 8 kHz

## wm8904 Driver

*kWM8904\_SampleRate12kHz* 11.025kHz, 12kHz  
*kWM8904\_SampleRate16kHz* 16kHz  
*kWM8904\_SampleRate24kHz* 22.05kHz, 24kHz  
*kWM8904\_SampleRate32kHz* 32kHz  
*kWM8904\_SampleRate48kHz* 44.1kHz, 48kHz

### 46.7.4.8 enum wm8904\_bit\_width\_t

Enumerator

*kWM8904\_BitWidth16* 16 bits  
*kWM8904\_BitWidth20* 20 bits  
*kWM8904\_BitWidth24* 24 bits  
*kWM8904\_BitWidth32* 32 bits

### 46.7.4.9 enum \_wm8904\_record\_source

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

### 46.7.4.10 enum \_wm8904\_record\_channel

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



#### 46.7.4.11 enum \_wm8904\_play\_source

Enumerator

*kWM8904\_PlaySourcePGA* play source PGA, bypass ADC  
*kWM8904\_PlaySourceDAC* play source Input3

#### 46.7.4.12 enum wm8904\_sys\_clk\_source\_t

Enumerator

*kWM8904\_SysClkSourceMCLK* wm8904 system clock soure from MCLK  
*kWM8904\_SysClkSourceFLL* wm8904 system clock soure from FLL

#### 46.7.4.13 enum wm8904\_fl\_clk\_source\_t

Enumerator

*kWM8904\_FLLClkSourceMCLK* wm8904 FLL clock source from MCLK

### 46.7.5 Function Documentation

#### 46.7.5.1 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.

#### 46.7.5.2 status\_t WM8904\_ReadRegister ( wm8904\_handle\_t \* *handle*, uint8\_t *reg*, uint16\_t \* *value* )

## wm8904 Driver

### Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8904 handle structure. |
| <i>reg</i>    | register address.        |
| <i>value</i>  | value to read.           |

### Returns

kStatus\_Success, else failed.

#### 46.7.5.3 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.

#### 46.7.5.4 status\_t WM8904\_Init ( wm8904\_handle\_t \* *handle*, wm8904\_config\_t \* *wm8904\_config* )

### Parameters

|                      |                                 |
|----------------------|---------------------------------|
| <i>handle</i>        | WM8904 handle structure.        |
| <i>wm8904_config</i> | WM8904 configuration structure. |

#### 46.7.5.5 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.

#### 46.7.5.6 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. |
|---------------|-----------------------------------|

#### 46.7.5.7 status\_t WM8904\_SetMasterSlave ( wm8904\_handle\_t \* *handle*, bool *master* )

**Deprecated** DO NOT USE THIS API ANYMORE. IT HAS BEEN SUPERCEDED BY [WM8904\\_SeMasterClock](#)

## 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.

#### 46.7.5.8 status\_t WM8904\_SeMasterClock ( wm8904\_handle\_t \* *handle*, uint32\_t *sysclk*, uint32\_t *sampleRate*, uint32\_t *bitWidth* )

## wm8904 Driver

### Parameters

|                   |                          |
|-------------------|--------------------------|
| <i>handle</i>     | WM8904 handle structure. |
| <i>sysclk</i>     | system clock rate.       |
| <i>sampleRate</i> | sample rate              |
| <i>bitWidth</i>   | bit width                |

### Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 46.7.5.9 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. |

#### 46.7.5.10 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.

#### 46.7.5.11 status\_t WM8904\_SetAudioFormat ( wm8904\_handle\_t \* *handle*, uint32\_t *sysclk*, uint32\_t *sampleRate*, uint32\_t *bitWidth* )

## Parameters

|                   |                                                                                                                                                                       |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>     | WM8904 handle structure.                                                                                                                                              |
| <i>sysclk</i>     | System clock frequency for codec, user should pay attention to this parater, sysclk is caculate as $SYSCLK = MCLK / MCLKDIV$ , MCLKDIV is bit0 of WM8904_CLK_RATES_0. |
| <i>sampleRate</i> | Sample rate frequency in Hz.                                                                                                                                          |
| <i>bitWidth</i>   | Audio data bit width.                                                                                                                                                 |

## Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 46.7.5.12 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.

#### 46.7.5.13 status\_t WM8904\_SetVolume ( wm8904\_handle\_t \* *handle*, uint16\_t *volumeLeft*, uint16\_t *volumeRight* )

The parameter should be from 0 to 100. The resulting volume will be. 0 for mute, 100 for maximum volume value.

## Parameters

## wm8904 Driver

|                    |                          |
|--------------------|--------------------------|
| <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.

### 46.7.5.14 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.

### 46.7.5.15 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.

### 46.7.5.16 status\_t WM8904\_EnableDACTDMMMode ( 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.

#### 46.7.5.17 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.

#### 46.7.5.18 status\_t WM8904\_SetModulePower ( wm8904\_handle\_t \* *handle*, wm8904\_module\_t *module*, bool *isEnabled* )

param handle WM8904 handle structure. param module wm8904 module. param isEnabled, true is power on, false is power down.

return kStatus\_WM8904\_Success if successful, different code otherwise..

#### 46.7.5.19 status\_t WM8904\_SetChannelVolume ( wm8904\_handle\_t \* *handle*, uint32\_t *channel*, uint32\_t *volume* )

The parameter should be from 0 to 100. The resulting volume will be. 0 for mute, 100 for maximum volume value.

## Parameters

## wm8904 Driver

|                |                         |
|----------------|-------------------------|
| <i>handle</i>  | codec handle structure. |
| <i>channel</i> | codec channel.          |
| <i>volume</i>  | volume value.           |

### Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 46.7.5.20 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_ModuleRecordSourceSingleEndMic, kCODEC_ModuleRecordSourceDigitalMic. |

### Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 46.7.5.21 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..

#### 46.7.5.22 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..

#### 46.7.5.23 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.

### 46.7.6 wm8904 adapter

#### 46.7.6.1 Overview

The wm8904 adapter provide codec unify control interface .

#### Macros

- #define [HAL\\_CODEC\\_HANDLER\\_SIZE](#) ([WM8904\\_I2C\\_HANDLER\\_SIZE](#) + 4)  
*codec handler size*

#### Enumerations

- enum [\\_codec\\_type](#) {  
    [kCODEC\\_CS42888](#),  
    [kCODEC\\_DA7212](#),  
    [kCODEC\\_WM8904](#),  
    [kCODEC\\_WM8960](#),  
    [kCODEC\\_WM8524](#),  
    [kCODEC\\_SGTL5000](#),  
    [kCODEC\\_DA7212](#),  
    [kCODEC\\_CS42888](#),  
    [kCODEC\\_AK4497](#),  
    [kCODEC\\_AK4458](#),  
    [kCODEC\\_TFA9XXX](#),  
    [kCODEC\\_TFA9896](#),  
    [kCODEC\\_SGTL5000](#),  
    [kCODEC\\_WM8904](#),  
    [kCODEC\\_WM8960](#) }  
*codec type*

#### Functions

- [status\\_t HAL\\_CODEC\\_Init](#) (void \*handle, void \*config)  
*Codec initialization.*
- [status\\_t HAL\\_CODEC\\_Deinit](#) (void \*handle)  
*Codec de-initialization.*
- [status\\_t HAL\\_CODEC\\_SetFormat](#) (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bit-Width)  
*set audio data format.*
- [status\\_t HAL\\_CODEC\\_SetVolume](#) (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- [status\\_t HAL\\_CODEC\\_SetMute](#) (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- [status\\_t HAL\\_CODEC\\_SetPower](#) (void \*handle, uint32\_t module, bool powerOn)

- *set audio codec module power.*  
**status\_t HAL\_CODEC\_SetRecord** (void \*handle, uint32\_t recordSource)
- *codec set record source.*  
**status\_t HAL\_CODEC\_SetRecordChannel** (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)
- *codec set record channel.*  
**status\_t HAL\_CODEC\_SetPlay** (void \*handle, uint32\_t playSource)
- *codec set play source.*  
**status\_t HAL\_CODEC\_ModuleControl** (void \*handle, uint32\_t cmd, uint32\_t data)
- *codec module control.*

## 46.7.6.2 Enumeration Type Documentation

### 46.7.6.2.1 enum\_codec\_type

Enumerator

**kCODEC\_CS42888** CS42888.  
**kCODEC\_DA7212** da7212  
**kCODEC\_WM8904** wm8904  
**kCODEC\_WM8960** wm8960  
**kCODEC\_WM8524** wm8524  
**kCODEC\_SGTL5000** sgtl5000  
**kCODEC\_DA7212** da7212  
**kCODEC\_CS42888** CS42888.  
**kCODEC\_AK4497** AK4497.  
**kCODEC\_AK4458** ak4458  
**kCODEC\_TFA9XXX** tfa9xxx  
**kCODEC\_TFA9896** tfa9896  
**kCODEC\_SGTL5000** sgtl5000  
**kCODEC\_WM8904** wm8904  
**kCODEC\_WM8960** wm8960

## 46.7.6.3 Function Documentation

### 46.7.6.3.1 status\_t HAL\_CODEC\_Init ( void \* handle, void \* config )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

## wm8904 Driver

|               |                      |
|---------------|----------------------|
| <i>config</i> | codec configuration. |
|---------------|----------------------|

Returns

kStatus\_Success is success, else initial failed.

### 46.7.6.3.2 status\_t HAL\_CODEC\_Deinit ( void \* *handle* )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

### 46.7.6.3.3 status\_t HAL\_CODEC\_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.

### 46.7.6.3.4 status\_t HAL\_CODEC\_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 <code>_codec_play_channel</code> . |
| <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.

#### 46.7.6.3.5 status\_t HAL\_CODEC\_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 <code>_codec_play_channel</code> . |
| <i>isMute</i>      | true is mute, false is unmute.                                                                  |

Returns

kStatus\_Success is success, else configure failed.

#### 46.7.6.3.6 status\_t HAL\_CODEC\_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.

#### 46.7.6.3.7 status\_t HAL\_CODEC\_SetRecord ( void \* *handle*, uint32\_t *recordSource* )

## wm8904 Driver

### 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.

#### 46.7.6.3.8 status\_t HAL\_CODEC\_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.

#### 46.7.6.3.9 status\_t HAL\_CODEC\_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.

#### 46.7.6.3.10 status\_t HAL\_CODEC\_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 <code>_codec_module_ctrl_cmd</code> .                                                                                                                                                                                                                               |
| <i>data</i>   | value to write, when cmd is <code>kCODEC_ModuleRecordSourceChannel</code> , the data should be a value combine of channel and source, please reference macro <code>CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN)</code> , reference codec specific driver for detail configurations. |

## Returns

`kStatus_Success` is success, else configure failed.





## Chapter 47

# Serial\_Manager

### 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\\_PORT\\_TYPE\\_UART](#) (0U)  
*Enable or disable uart port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_USBCDC](#) (0U)  
*Enable or disable USB CDC port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_SWO](#) (0U)  
*Enable or disable SWO port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_USBCDC\\_VIRTUAL](#) (0U)  
*Enable or disable USB CDC virtual port (1 - enable, 0 - disable)*
- #define [SERIAL\\_MANAGER\\_WRITE\\_HANDLE\\_SIZE](#) (4U)  
*Set serial manager write handle size.*
- #define [SERIAL\\_MANAGER\\_HANDLE\\_SIZE](#) (SERIAL\_MANAGER\_HANDLE\_SIZE\_TEMP + 12U)  
*SERIAL\_PORT\_UART\_HANDLE\_SIZE/SERIAL\_PORT\_USB\_CDC\_HANDLE\_SIZE + serial manager dedicated size.*
- #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.*

## Overview

- #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\\_USE\\_COMMON\\_TASK](#) (1U)  
*Macro to determine whether use common task.*
- #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)  
*callback function*

## Enumerations

- enum [serial\\_port\\_type\\_t](#) {  
    [kSerialPort\\_Uart](#) = 1U,  
    [kSerialPort\\_UsbCdc](#),  
    [kSerialPort\\_Swo](#),  
    [kSerialPort\\_UsbCdcVirtual](#) }  
*serial port type*
- enum [serial\\_manager\\_status\\_t](#) {  
    [kStatus\\_SerialManager\\_Success](#) = kStatus\_Success,  
    [kStatus\\_SerialManager\\_Error](#) = MAKE\_STATUS(kStatusGroup\_SERIALMANAGER, 1),  
    [kStatus\\_SerialManager\\_Busy](#) = MAKE\_STATUS(kStatusGroup\_SERIALMANAGER, 2),  
    [kStatus\\_SerialManager\\_Notify](#) = MAKE\_STATUS(kStatusGroup\_SERIALMANAGER, 3),  
    [kStatus\\_SerialManager\\_Canceled](#),  
    [kStatus\\_SerialManager\\_HandleConflict](#) = MAKE\_STATUS(kStatusGroup\_SERIALMANAGER, 5),  
    [kStatus\\_SerialManager\\_RingBufferOverflow](#),  
    [kStatus\\_SerialManager\\_NotConnected](#) = MAKE\_STATUS(kStatusGroup\_SERIALMANAGER, 7) }  
*serial manager error code*

## Functions

- [serial\\_manager\\_status\\_t](#) [SerialManager\\_Init](#) ([serial\\_handle\\_t](#) serialHandle, [serial\\_manager\\_config\\_t](#) \*config)  
*Initializes a serial manager module with the serial manager handle and the user configuration structure.*

- [serial\\_manager\\_status\\_t SerialManager\\_Deinit](#) ([serial\\_handle\\_t](#) serialHandle)  
*De-initializes the serial manager module instance.*
- [serial\\_manager\\_status\\_t SerialManager\\_OpenWriteHandle](#) ([serial\\_handle\\_t](#) serialHandle, [serial\\_write\\_handle\\_t](#) writeHandle)  
*Opens a writing handle for the serial manager module.*
- [serial\\_manager\\_status\\_t SerialManager\\_CloseWriteHandle](#) ([serial\\_write\\_handle\\_t](#) writeHandle)  
*Closes a writing handle for the serial manager module.*
- [serial\\_manager\\_status\\_t SerialManager\\_OpenReadHandle](#) ([serial\\_handle\\_t](#) serialHandle, [serial\\_read\\_handle\\_t](#) readHandle)  
*Opens a reading handle for the serial manager module.*
- [serial\\_manager\\_status\\_t SerialManager\\_CloseReadHandle](#) ([serial\\_read\\_handle\\_t](#) readHandle)  
*Closes a reading for the serial manager module.*
- [serial\\_manager\\_status\\_t SerialManager\\_WriteBlocking](#) ([serial\\_write\\_handle\\_t](#) writeHandle, [uint8\\_t](#) \*buffer, [uint32\\_t](#) length)  
*Transmits data with the blocking mode.*
- [serial\\_manager\\_status\\_t SerialManager\\_ReadBlocking](#) ([serial\\_read\\_handle\\_t](#) readHandle, [uint8\\_t](#) \*buffer, [uint32\\_t](#) length)  
*Reads data with the blocking mode.*
- [serial\\_manager\\_status\\_t SerialManager\\_EnterLowpower](#) ([serial\\_handle\\_t](#) serialHandle)  
*Prepares to enter low power consumption.*
- [serial\\_manager\\_status\\_t SerialManager\\_ExitLowpower](#) ([serial\\_handle\\_t](#) serialHandle)  
*Restores from low power consumption.*

## Data Structure Documentation

### 47.2.1 struct serial\_manager\_config\_t

#### Data Fields

- [serial\\_port\\_type\\_t](#) type  
*Serial port type.*
- void \* [portConfig](#)  
*Serial port configuration.*

### 47.2.2 struct serial\_manager\_callback\_message\_t

#### Data Fields

- [uint8\\_t](#) \* [buffer](#)  
*Transferred buffer.*
- [uint32\\_t](#) [length](#)  
*Transferred data length.*

Macro Definition Documentation

47.3.1 **#define SERIAL\_MANAGER\_HANDLE\_SIZE (SERIAL\_MANAGER\_HANDLE\_SIZE\_TEMP + 12U)**

Definition of serial manager handle size.

47.3.2 **#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. |
|-------------|-----------------------------------------------|

47.3.3 **#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. |
|-------------|-----------------------------------------------------|

**47.3.4 #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\_<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. |
|-------------|----------------------------------------------------|

**47.3.5 #define SERIAL\_MANAGER\_USE\_COMMON\_TASK (1U)**

**47.3.6 #define SERIAL\_MANAGER\_TASK\_PRIORITY (2U)**

**47.3.7 #define SERIAL\_MANAGER\_TASK\_STACK\_SIZE (1000U)**

## Enumeration Type Documentation

**47.4.1 enum serial\_port\_type\_t**

### Enumerator

*kSerialPort\_Uart* Serial port UART.

*kSerialPort\_UsbCdc* Serial port USB CDC.

*kSerialPort\_Swo* Serial port SWO.

*kSerialPort\_UsbCdcVirtual* Serial port USB CDC Virtual.

## Function Documentation

### 47.4.2 enum serial\_manager\_status\_t

Enumerator

*kStatus\_SerialManager\_Success* Success.  
*kStatus\_SerialManager\_Error* Failed.  
*kStatus\_SerialManager\_Busy* Busy.  
*kStatus\_SerialManager\_Notify* Ring buffer is not empty.  
*kStatus\_SerialManager\_Canceled* the non-blocking request is canceled  
*kStatus\_SerialManager\_HandleConflict* The handle is opened.  
*kStatus\_SerialManager\_RingBufferOverflow* The ring buffer is overflowed.  
*kStatus\_SerialManager\_NotConnected* The host is not connected.

## Function Documentation

### 47.5.1 serial\_manager\_status\_t SerialManager\_Init ( serial\_handle\_t serialHandle, serial\_manager\_config\_t \* config )

This function configures the Serial Manager module with user-defined settings. The user can configure the configuration structure. The parameter serialHandle is a pointer to point to a memory space of size [SERIAL\\_MANAGER\\_HANDLE\\_SIZE](#) allocated by the caller. The Serial Manager module supports three types of serial port, UART (includes UART, USART, LPUART, etc), USB CDC and sww. 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;
* 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 <a href="#">SERIAL_MANAGER_HANDLE_SIZE</a> 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: <a href="#">SERIAL_MANAGER_HANDLE_DEFINE(serialHandle)</a> ; 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. |

### 47.5.2 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. |
|--------------------------------------|-----------------------------------------------|

## Function Documentation

|                                   |                                                 |
|-----------------------------------|-------------------------------------------------|
| <i>kStatus_SerialManager_Busy</i> | Opened reading or writing handle is not closed. |
|-----------------------------------|-------------------------------------------------|

### 47.5.3 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: <a href="#">SERIAL_MANAGER_WRITE_HANDLE_DEFINE(writeHandle)</a> ; 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);
*

```

#### 47.5.4 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. |
|--------------------------------------|-------------------------------|

#### 47.5.5 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: <code>SERIAL_MANAGER_READ_HANDLE_DEFINE(readHandle);</code> or <code>uint32_t readHandle[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code> |

## Function Documentation

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));
*
```

### 47.5.6 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. |
|--------------------------------------|-------------------------------|

### 47.5.7 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_-<br/>Success</i> | Successfully sent all data.                                      |
| <i>kStatus_SerialManager_-<br/>Busy</i>    | Previous transmission still not finished; data not all sent yet. |
| <i>kStatus_SerialManager_-<br/>Error</i>   | An error occurred.                                               |

#### 47.5.8 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

## Function Documentation

|                   |                                                       |
|-------------------|-------------------------------------------------------|
| <i>readHandle</i> | The serial manager module handle pointer.             |
| <i>buffer</i>     | Start address of the data to store the received data. |
| <i>length</i>     | The length of the data to be received.                |

Return values

|                                            |                                                                      |
|--------------------------------------------|----------------------------------------------------------------------|
| <i>kStatus_SerialManager_-<br/>Success</i> | Successfully received all data.                                      |
| <i>kStatus_SerialManager_-<br/>Busy</i>    | Previous transmission still not finished; data not all received yet. |
| <i>kStatus_SerialManager_-<br/>Error</i>   | An error occurred.                                                   |

### 47.5.9 serial\_manager\_status\_t SerialManager\_EnterLowpower ( serial\_handle\_t serialHandle )

This function is used to prepare to enter low power consumption.

Parameters

|                     |                                           |
|---------------------|-------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. |
|---------------------|-------------------------------------------|

Return values

|                                            |                       |
|--------------------------------------------|-----------------------|
| <i>kStatus_SerialManager_-<br/>Success</i> | Successful operation. |
|--------------------------------------------|-----------------------|

### 47.5.10 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_-<br/>Success</i> | Successful operation. |
|--------------------------------------------|-----------------------|

## Serial Port Uart

## Serial Port Uart

### 47.6.1 Overview

#### Data Structures

- struct [serial\\_port\\_uart\\_config\\_t](#)  
*serial port uart config struct [More...](#)*

#### Macros

- #define [SERIAL\\_PORT\\_UART\\_HANDLE\\_SIZE](#) (HAL\_UART\_HANDLE\_SIZE)  
*serial port uart handle size*

#### Enumerations

- enum [serial\\_port\\_uart\\_parity\\_mode\\_t](#) {  
    [kSerialManager\\_UartParityDisabled](#) = 0x0U,  
    [kSerialManager\\_UartParityEven](#) = 0x1U,  
    [kSerialManager\\_UartParityOdd](#) = 0x2U }  
*serial port uart parity mode*
- enum [serial\\_port\\_uart\\_stop\\_bit\\_count\\_t](#) {  
    [kSerialManager\\_UartOneStopBit](#) = 0U,  
    [kSerialManager\\_UartTwoStopBit](#) = 1U }  
*serial port uart stop bit count*

### 47.6.2 Data Structure Documentation

#### 47.6.2.1 struct serial\_port\_uart\_config\_t

##### Data Fields

- uint32\_t [clockRate](#)  
*clock rate*
- uint32\_t [baudRate](#)  
*baud rate*
- [serial\\_port\\_uart\\_parity\\_mode\\_t](#) [parityMode](#)  
*Parity mode, disabled (default), even, odd.*
- [serial\\_port\\_uart\\_stop\\_bit\\_count\\_t](#) [stopBitCount](#)  
*Number of stop bits, 1 stop bit (default) or 2 stop bits.*
- uint8\_t [instance](#)  
*Instance (0 - UART0, 1 - UART1, ...), detail information please refer to the SOC corresponding RM.*
- uint8\_t [enableRx](#)  
*Enable RX.*
- uint8\_t [enableTx](#)

*Enable TX.*

#### 47.6.2.1.0.1 Field Documentation

47.6.2.1.0.1.1 `uint8_t serial_port_uart_config_t::instance`

### 47.6.3 Enumeration Type Documentation

#### 47.6.3.1 `enum serial_port_uart_parity_mode_t`

Enumerator

*kSerialManager\_UartParityDisabled* Parity disabled.  
*kSerialManager\_UartParityEven* Parity even enabled.  
*kSerialManager\_UartParityOdd* Parity odd enabled.

#### 47.6.3.2 `enum serial_port_uart_stop_bit_count_t`

Enumerator

*kSerialManager\_UartOneStopBit* One stop bit.  
*kSerialManager\_UartTwoStopBit* Two stop bits.

## Serial Port USB

### 47.7.1 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](#) (72)  
*serial port usb handle size*
- #define [USB\\_DEVICE\\_INTERRUPT\\_PRIORITY](#) (3U)  
*USB interrupt priority.*

#### Enumerations

- enum [serial\\_port\\_usb\\_cdc\\_controller\\_index\\_t](#) {  
    [kSerialManager\\_UsbControllerKhci0](#) = 0U,  
    [kSerialManager\\_UsbControllerKhci1](#) = 1U,  
    [kSerialManager\\_UsbControllerEhci0](#) = 2U,  
    [kSerialManager\\_UsbControllerEhci1](#) = 3U,  
    [kSerialManager\\_UsbControllerLpcIp3511Fs0](#) = 4U,  
    [kSerialManager\\_UsbControllerLpcIp3511Fs1](#) = 5U,  
    [kSerialManager\\_UsbControllerLpcIp3511Hs0](#) = 6U,  
    [kSerialManager\\_UsbControllerLpcIp3511Hs1](#) = 7U,  
    [kSerialManager\\_UsbControllerOhci0](#) = 8U,  
    [kSerialManager\\_UsbControllerOhci1](#) = 9U,  
    [kSerialManager\\_UsbControllerIp3516Hs0](#) = 10U,  
    [kSerialManager\\_UsbControllerIp3516Hs1](#) = 11U }  
*USB controller ID.*



## 47.7.2 Data Structure Documentation

### 47.7.2.1 struct serial\_port\_usb\_cdc\_config\_t

#### Data Fields

- [serial\\_port\\_usb\\_cdc\\_controller\\_index\\_t](#) controllerIndex  
controller index

## 47.7.3 Enumeration Type Documentation

### 47.7.3.1 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.

#### **47.7.4 USB Device Configuration**

## Serial Port SWO

### 47.8.1 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*

### 47.8.2 Data Structure Documentation

#### 47.8.2.1 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.*

### 47.8.3 Enumeration Type Documentation

#### 47.8.3.1 enum serial\_port\_swo\_protocol\_t

##### Enumerator

***kSerialManager\_SwoProtocolManchester*** SWO Manchester protocol.  
***kSerialManager\_SwoProtocolNrz*** SWO UART/NRZ protocol.



## Chapter 48

### Data Structure Documentation

#### 48.0.4 codec\_i2c\_config\_t Struct Reference

CODEC I2C configurations structure.

```
#include <fsl_codec_i2c.h>
```

##### Data Fields

- uint32\_t [codecI2CInstance](#)  
*i2c bus instance*
- uint32\_t [codecI2CSourceClock](#)  
*i2c bus source clock frequency*



***How to Reach Us:*****Home Page:**[nxp.com](http://nxp.com)**Web Support:**[nxp.com/support](http://nxp.com/support)

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

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

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, Freescale, the Freescale logo, Kinetis, Processor Expert, and Tower are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, Cortex, Keil, Mbed, Mbed Enabled, and Vision are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2018 NXP B.V.