
Document Number: MCUXSDKAPIRM
Rev 2.14.0
Jul 2023

MCUXpresso SDK API Reference Manual

NXP Semiconductors



Contents

Chapter 1 Introduction

Chapter 2 Trademarks

Chapter 3 Architectural Overview

Chapter 4 Clock Driver

4.1	Overview	7
4.2	Data Structure Documentation	18
4.2.1	struct clock_usb_pll_config_t	18
4.2.2	struct clock_sys_pll_config_t	18
4.2.3	struct clock_audio_pll_config_t	19
4.2.4	struct clock_enet_pll_config_t	19
4.3	Macro Definition Documentation	20
4.3.1	FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL	20
4.3.2	FSL_CLOCK_DRIVER_VERSION	20
4.3.3	ADC_CLOCKS	20
4.3.4	AOI_CLOCKS	21
4.3.5	BEE_CLOCKS	21
4.3.6	CMP_CLOCKS	21
4.3.7	DCDC_CLOCKS	21
4.3.8	DCP_CLOCKS	21
4.3.9	DMAMUX_CLOCKS	22
4.3.10	EDMA_CLOCKS	22
4.3.11	ENC_CLOCKS	22
4.3.12	ENET_CLOCKS	22
4.3.13	EWM_CLOCKS	22
4.3.14	FLEXCAN_CLOCKS	23
4.3.15	FLEXCAN_PERIPH_CLOCKS	23
4.3.16	FLEXIO_CLOCKS	23
4.3.17	FLEXRAM_CLOCKS	23
4.3.18	FLEXSPI_CLOCKS	23
4.3.19	FLEXSPI_EXSC_CLOCKS	24
4.3.20	GPIO_CLOCKS	24
4.3.21	GPT_CLOCKS	24

Section No.	Title	Page No.
4.3.22	KPP_CLOCKS	24
4.3.23	LPI2C_CLOCKS	24
4.3.24	LPSPI_CLOCKS	25
4.3.25	LPUART_CLOCKS	25
4.3.26	OCRAM_EXSC_CLOCKS	25
4.3.27	PIT_CLOCKS	25
4.3.28	PWM_CLOCKS	25
4.3.29	RTWDOG_CLOCKS	26
4.3.30	SAI_CLOCKS	26
4.3.31	SEMC_CLOCKS	26
4.3.32	SEMC_EXSC_CLOCKS	26
4.3.33	TMR_CLOCKS	27
4.3.34	TRNG_CLOCKS	27
4.3.35	WDOG_CLOCKS	27
4.3.36	USDHC_CLOCKS	27
4.3.37	SPDIF_CLOCKS	27
4.3.38	XBARA_CLOCKS	28
4.3.39	XBARB_CLOCKS	28
4.3.40	kCLOCK_CoreSysClk	28
4.3.41	CLOCK_GetCoreSysClkFreq	28
4.4	Enumeration Type Documentation	28
4.4.1	clock_name_t	28
4.4.2	clock_ip_name_t	29
4.4.3	clock_osc_t	31
4.4.4	clock_gate_value_t	31
4.4.5	clock_mode_t	31
4.4.6	clock_mux_t	32
4.4.7	clock_div_t	32
4.4.8	clock_div_value_t	33
4.4.9	clock_usb_src_t	38
4.4.10	clock_usb_phy_src_t	38
4.4.11	_clock_pll_clk_src	38
4.4.12	clock_pll_t	38
4.4.13	clock_pfd_t	38
4.4.14	clock_output1_selection_t	39
4.4.15	clock_output2_selection_t	39
4.4.16	clock_output_divider_t	39
4.4.17	clock_root_t	40
4.5	Function Documentation	40
4.5.1	CLOCK_SetMux	40
4.5.2	CLOCK_GetMux	40
4.5.3	CLOCK_SetDiv	41
4.5.4	CLOCK_GetDiv	41

Section No.	Title	Page No.
4.5.5	CLOCK_ControlGate	41
4.5.6	CLOCK_EnableClock	41
4.5.7	CLOCK_DisableClock	42
4.5.8	CLOCK_SetMode	42
4.5.9	CLOCK_GetOscFreq	42
4.5.10	CLOCK_GetAhbFreq	42
4.5.11	CLOCK_GetSemcFreq	42
4.5.12	CLOCK_GetIpgFreq	43
4.5.13	CLOCK_GetPerClkFreq	43
4.5.14	CLOCK_GetFreq	43
4.5.15	CLOCK_GetCpuClkFreq	43
4.5.16	CLOCK_GetClockRootFreq	43
4.5.17	CLOCK_InitExternalClk	44
4.5.18	CLOCK_DeinitExternalClk	44
4.5.19	CLOCK_SwitchOsc	44
4.5.20	CLOCK_GetRtcFreq	44
4.5.21	CLOCK_SetXtalFreq	44
4.5.22	CLOCK_SetRtcXtalFreq	45
4.5.23	CLOCK_EnableUsbhs0Clock	45
4.6	Variable Documentation	45
4.6.1	g_xtalFreq	45
4.6.2	g_rtcXtalFreq	45
 Chapter 5 ROMAPI Driver		
5.1	Overview	46
5.2	Data Structure Documentation	48
5.2.1	struct flexspi_lut_seq_t	48
5.2.2	struct flexspi_mem_config_t	49
5.2.3	struct flexspi_nor_config_t	51
5.2.4	struct flexspi_xfer_t	51
5.3	Macro Definition Documentation	52
5.3.1	FSL_ROM_ROMAPI_VERSION	52
5.3.2	FSL_ROM_FLEXSPINOR_DRIVER_VERSION	52
5.3.3	kROM_StatusGroup_FLEXSPI	52
5.3.4	kROM_StatusGroup_FLEXSPINOR	52
5.3.5	FLEXSPI_CFG_BLK_TAG	52
5.3.6	NOR_CMD_LUT_SEQ_IDX_READ	52
5.4	Enumeration Type Documentation	52
5.4.1	anonymous enum	52
5.4.2	anonymous enum	52

Section No.	Title	Page No.
5.4.3	anonymous enum	53
5.4.4	anonymous enum	53
5.4.5	anonymous enum	53
5.4.6	_flexspi_nor_status	53
5.4.7	flexspi_operation_t	54
Chapter 6 IOMUXC: IOMUX Controller		
6.1	Overview	55
6.2	Macro Definition Documentation	77
6.2.1	FSL_IOMUXC_DRIVER_VERSION	77
6.3	Function Documentation	77
6.3.1	IOMUXC_SetPinMux	77
6.3.2	IOMUXC_SetPinConfig	78
6.3.3	IOMUXC_EnableMode	78
6.3.4	IOMUXC_SetSaiMClkClockSource	79
6.3.5	IOMUXC_MQSEnterSoftwareReset	79
6.3.6	IOMUXC_MQSEnable	79
6.3.7	IOMUXC_MQSConfig	79
Chapter 7 ADC: 12-bit Analog to Digital Converter Driver		
7.1	Overview	81
7.2	Typical use case	81
7.2.1	Polling Configuration	81
7.2.2	Polling Configuration	81
7.3	Data Structure Documentation	83
7.3.1	struct adc_config_t	83
7.3.2	struct adc_offest_config_t	84
7.3.3	struct adc_hardware_compare_config_t	85
7.3.4	struct adc_channel_config_t	85
7.4	Macro Definition Documentation	86
7.4.1	FSL_ADC_DRIVER_VERSION	86
7.5	Enumeration Type Documentation	86
7.5.1	adc_status_flags_t	86
7.5.2	adc_reference_voltage_source_t	86
7.5.3	adc_sample_period_mode_t	86
7.5.4	adc_clock_source_t	87
7.5.5	adc_clock_driver_t	87
7.5.6	adc_resolution_t	87

Section No.	Title	Page No.
7.5.7	adc_hardware_compare_mode_t	87
7.5.8	adc_hardware_average_mode_t	88
7.6	Function Documentation	88
7.6.1	ADC_Init	88
7.6.2	ADC_Deinit	88
7.6.3	ADC_GetDefaultConfig	88
7.6.4	ADC_SetChannelConfig	89
7.6.5	ADC_GetChannelConversionValue	89
7.6.6	ADC_GetChannelStatusFlags	90
7.6.7	ADC_DoAutoCalibration	90
7.6.8	ADC_SetOffsetConfig	91
7.6.9	ADC_EnableDMA	91
7.6.10	ADC_EnableHardwareTrigger	91
7.6.11	ADC_SetHardwareCompareConfig	91
7.6.12	ADC_SetHardwareAverageConfig	92
7.6.13	ADC_GetStatusFlags	92
7.6.14	ADC_ClearStatusFlags	92
Chapter 8 ADC_ETC: ADC External Trigger Control		
8.1	Overview	93
8.2	Typical use case	93
8.2.1	Software trigger Configuration	93
8.2.2	Hardware trigger Configuration	93
8.3	Data Structure Documentation	94
8.3.1	struct adc_etc_config_t	94
8.3.2	struct adc_etc_trigger_chain_config_t	94
8.3.3	struct adc_etc_trigger_config_t	94
8.4	Macro Definition Documentation	94
8.4.1	FSL_ADC_ETC_DRIVER_VERSION	94
8.4.2	ADC_ETC_DMA_CTRL_TRGn_REQ_MASK	95
8.5	Function Documentation	95
8.5.1	ADC_ETC_Init	95
8.5.2	ADC_ETC_Deinit	95
8.5.3	ADC_ETC_GetDefaultConfig	95
8.5.4	ADC_ETC_SetTriggerConfig	95
8.5.5	ADC_ETC_SetTriggerChainConfig	96
8.5.6	ADC_ETC_GetInterruptStatusFlags	96
8.5.7	ADC_ETC_ClearInterruptStatusFlags	96
8.5.8	ADC_ETC_EnableDMA	97

Section No.	Title	Page No.
8.5.9	ADC_ETC_DisableDMA	97
8.5.10	ADC_ETC_GetDMAStatusFlags	97
8.5.11	ADC_ETC_ClearDMAStatusFlags	98
8.5.12	ADC_ETC_DoSoftwareReset	99
8.5.13	ADC_ETC_DoSoftwareTrigger	99
8.5.14	ADC_ETC_GetADCCConversionValue	99

Chapter 9 AIPSTZ: AHB to IP Bridge

9.1	Overview	101
9.2	Enumeration Type Documentation	101
9.2.1	aipstz_master_privilege_level_t	101
9.2.2	aipstz_master_t	102
9.2.3	aipstz_peripheral_access_control_t	102
9.2.4	aipstz_peripheral_t	102
9.3	Function Documentation	102
9.3.1	AIPSTZ_SetMasterPriviledgeLevel	102
9.3.2	AIPSTZ_SetPeripheralAccessControl	102

Chapter 10 AOI: Crossbar AND/OR/INVERT Driver

10.1	Overview	104
10.2	Function groups	104
10.2.1	AOI Initialization	104
10.2.2	AOI Get Set Operation	104
10.3	Typical use case	104
10.4	Data Structure Documentation	106
10.4.1	struct aoi_event_config_t	106
10.5	Macro Definition Documentation	107
10.5.1	FSL_AOI_DRIVER_VERSION	107
10.6	Enumeration Type Documentation	107
10.6.1	aoi_input_config_t	107
10.6.2	aoi_event_t	107
10.7	Function Documentation	107
10.7.1	AOI_Init	107
10.7.2	AOI_Deinit	107
10.7.3	AOI_GetEventLogicConfig	108
10.7.4	AOI_SetEventLogicConfig	108

Section No.	Title	Page No.
Chapter 11 BEE: Bus Encryption Engine		
11.1	Overview	110
11.2	BEE Driver Initialization and Configuration	110
11.3	Enable & Disable BEE	110
11.4	Set BEE region config and key	110
11.5	Status	111
11.5.1	BEE example	111
11.6	Data Structure Documentation	112
11.6.1	struct bee_region_config_t	112
11.7	Macro Definition Documentation	113
11.7.1	FSL_BEE_DRIVER_VERSION	113
11.8	Enumeration Type Documentation	114
11.8.1	bee_aes_mode_t	114
11.8.2	bee_region_t	114
11.8.3	bee_ac_prot_enable	114
11.8.4	bee_endian_swap_enable	114
11.8.5	bee_security_level	114
11.8.6	bee_status_flags_t	115
11.9	Function Documentation	115
11.9.1	BEE_Init	115
11.9.2	BEE_Deinit	115
11.9.3	BEE_Enable	115
11.9.4	BEE_Disable	116
11.9.5	BEE_GetDefaultConfig	116
11.9.6	BEE_SetConfig	116
11.9.7	BEE_SetRegionKey	117
11.9.8	BEE_SetRegionNonce	117
11.9.9	BEE_GetStatusFlags	117
11.9.10	BEE_ClearStatusFlags	118
Chapter 12 CACHE: ARMV7-M7 CACHE Memory Controller		
12.1	Overview	119
12.2	Function groups	119
12.2.1	L1 CACHE Operation	119
12.2.2	L2 CACHE Operation	119

Section No.	Title	Page No.
12.3	Macro Definition Documentation	120
12.3.1	FSL_CACHE_DRIVER_VERSION	120
12.4	Function Documentation	121
12.4.1	L1CACHE_InvalidateICacheByRange	121
12.4.2	L1CACHE_InvalidateDCacheByRange	122
12.4.3	L1CACHE_CleanDCacheByRange	122
12.4.4	L1CACHE_CleanInvalidateDCacheByRange	123
12.4.5	ICACHE_InvalidateByRange	124
12.4.6	DCACHE_InvalidateByRange	124
12.4.7	DCACHE_CleanByRange	125
12.4.8	DCACHE_CleanInvalidateByRange	126
Chapter 13 CMP: Analog Comparator Driver		
13.1	Overview	127
13.2	Typical use case	127
13.2.1	Polling Configuration	127
13.2.2	Interrupt Configuration	127
13.3	Data Structure Documentation	129
13.3.1	struct cmp_config_t	129
13.3.2	struct cmp_filter_config_t	129
13.3.3	struct cmp_dac_config_t	130
13.4	Macro Definition Documentation	130
13.4.1	FSL_CMP_DRIVER_VERSION	130
13.5	Enumeration Type Documentation	130
13.5.1	_cmp_interrupt_enable	130
13.5.2	_cmp_status_flags	130
13.5.3	cmp_hysteresis_mode_t	131
13.5.4	cmp_reference_voltage_source_t	131
13.6	Function Documentation	131
13.6.1	CMP_Init	131
13.6.2	CMP_Deinit	131
13.6.3	CMP_Enable	132
13.6.4	CMP_GetDefaultConfig	132
13.6.5	CMP_SetInputChannels	132
13.6.6	CMP_SetFilterConfig	133
13.6.7	CMP_SetDACConfig	133
13.6.8	CMP_EnableInterrupts	133
13.6.9	CMP_DisableInterrupts	133
13.6.10	CMP_GetStatusFlags	134

Section No.	Title	Page No.
13.6.11	CMP_ClearStatusFlags	134
Chapter 14 Common Driver		
14.1	Overview	135
14.2	Macro Definition Documentation	138
14.2.1	FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ	138
14.2.2	MAKE_STATUS	138
14.2.3	MAKE_VERSION	138
14.2.4	FSL_COMMON_DRIVER_VERSION	139
14.2.5	DEBUG_CONSOLE_DEVICE_TYPE_NONE	139
14.2.6	DEBUG_CONSOLE_DEVICE_TYPE_UART	139
14.2.7	DEBUG_CONSOLE_DEVICE_TYPE_LPUART	139
14.2.8	DEBUG_CONSOLE_DEVICE_TYPE_LPSCI	139
14.2.9	DEBUG_CONSOLE_DEVICE_TYPE_USBCDC	139
14.2.10	DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM	139
14.2.11	DEBUG_CONSOLE_DEVICE_TYPE_IUART	139
14.2.12	DEBUG_CONSOLE_DEVICE_TYPE_VUSART	139
14.2.13	DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART	139
14.2.14	DEBUG_CONSOLE_DEVICE_TYPE_SWO	139
14.2.15	DEBUG_CONSOLE_DEVICE_TYPE_QSCI	139
14.2.16	ARRAY_SIZE	139
14.3	Typedef Documentation	139
14.3.1	status_t	139
14.4	Enumeration Type Documentation	139
14.4.1	_status_groups	139
14.4.2	anonymous enum	142
14.5	Function Documentation	142
14.5.1	SDK_Malloc	142
14.5.2	SDK_Free	143
14.5.3	SDK_DelayAtLeastUs	143
Chapter 15 DCDC: DCDC Converter		
15.1	Overview	144
15.2	Function groups	144
15.2.1	Initialization and deinitialization	144
15.2.2	Status	144
15.2.3	Misc control	144
15.3	Application guideline	145

Section No.	Title	Page No.
15.3.1	Continuous conduction mode	145
15.3.2	Discontinuous conduction mode	145
15.4	Data Structure Documentation	147
15.4.1	struct dcdc_detection_config_t	147
15.4.2	struct dcdc_loop_control_config_t	148
15.4.3	struct dcdc_low_power_config_t	149
15.4.4	struct dcdc_internal_regulator_config_t	150
15.4.5	struct dcdc_min_power_config_t	150
15.5	Macro Definition Documentation	150
15.5.1	FSL_DCDC_DRIVER_VERSION	150
15.6	Enumeration Type Documentation	151
15.6.1	_dcdc_status_flags_t	151
15.6.2	dcdc_comparator_current_bias_t	151
15.6.3	dcdc_over_current_threshold_t	151
15.6.4	dcdc_peak_current_threshold_t	151
15.6.5	dcdc_count_charging_time_period_t	152
15.6.6	dcdc_count_charging_time_threshold_t	152
15.6.7	dcdc_clock_source_t	152
15.7	Function Documentation	152
15.7.1	DCDC_Init	152
15.7.2	DCDC_Deinit	152
15.7.3	DCDC_GetstatusFlags	153
15.7.4	DCDC_EnableOutputRangeComparator	153
15.7.5	DCDC_SetClockSource	153
15.7.6	DCDC_GetDefaultDetectionConfig	153
15.7.7	DCDC_SetDetectionConfig	154
15.7.8	DCDC_GetDefaultLowPowerConfig	154
15.7.9	DCDC_SetLowPowerConfig	154
15.7.10	DCDC_ResetCurrentAlertSignal	155
15.7.11	DCDC_SetBandgapVoltageTrimValue	155
15.7.12	DCDC_GetDefaultLoopControlConfig	155
15.7.13	DCDC_SetLoopControlConfig	156
15.7.14	DCDC_SetMinPowerConfig	156
15.7.15	DCDC_SetLPComparatorBiasValue	156
15.7.16	DCDC_LockTargetVoltage	156
15.7.17	DCDC_AdjustTargetVoltage	157
15.7.18	DCDC_AdjustRunTargetVoltage	157
15.7.19	DCDC_AdjustLowPowerTargetVoltage	158
15.7.20	DCDC_SetInternalRegulatorConfig	158
15.7.21	DCDC_EnableImproveTransition	158
15.7.22	DCDC_BootIntoDCM	158

Section No.	Title	Page No.
15.7.23	DCDC_BootIntoCCM	159
Chapter 16 DCP: Data Co-Processor		
16.1	Overview	160
16.2	DCP Driver Initialization and Configuration	160
16.3	Comments about API usage in RTOS	161
16.4	Comments about API usage in interrupt handler	161
16.5	Comments about DCACHE	161
16.6	DCP Driver Examples	161
16.6.1	Simple examples	161
16.7	Data Structure Documentation	163
16.7.1	struct dcp_work_packet_t	163
16.7.2	struct dcp_handle_t	163
16.7.3	struct dcp_context_t	163
16.7.4	struct dcp_config_t	163
16.8	Macro Definition Documentation	164
16.8.1	FSL_DCP_DRIVER_VERSION	164
16.9	Enumeration Type Documentation	165
16.9.1	_dcp_status	165
16.9.2	_dcp_ch_enable_t	165
16.9.3	_dcp_ch_int_enable_t	165
16.9.4	dcp_channel_t	165
16.9.5	dcp_key_slot_t	166
16.9.6	dcp_swap_t	166
16.10	Function Documentation	166
16.10.1	DCP_Init	166
16.10.2	DCP_Deinit	166
16.10.3	DCP_GetDefaultConfig	166
16.10.4	DCP_WaitForChannelComplete	167
16.11	DCP AES blocking driver	168
16.11.1	Overview	168
16.11.2	Function Documentation	168
16.12	DCP AES non-blocking driver	172
16.12.1	Overview	172
16.12.2	Function Documentation	172

Section No.	Title	Page No.
16.13	DCP HASH driver	176
16.13.1	Overview	176
16.13.2	Data Structure Documentation	177
16.13.3	Macro Definition Documentation	177
16.13.4	Enumeration Type Documentation	177
16.13.5	Function Documentation	177
Chapter 17 DMAMUX: Direct Memory Access Multiplexer Driver		
17.1	Overview	180
17.2	Typical use case	180
17.2.1	DMAMUX Operation	180
17.3	Macro Definition Documentation	180
17.3.1	FSL_DMAMUX_DRIVER_VERSION	180
17.4	Function Documentation	180
17.4.1	DMAMUX_Init	180
17.4.2	DMAMUX_Deinit	181
17.4.3	DMAMUX_EnableChannel	181
17.4.4	DMAMUX_DisableChannel	181
17.4.5	DMAMUX_SetSource	182
Chapter 18 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver		
18.1	Overview	183
18.2	Typical use case	183
18.2.1	eDMA Operation	183
18.3	Data Structure Documentation	188
18.3.1	struct edma_config_t	188
18.3.2	struct edma_transfer_config_t	189
18.3.3	struct edma_channel_Preemption_config_t	190
18.3.4	struct edma_minor_offset_config_t	190
18.3.5	struct edma_tcd_t	191
18.3.6	struct edma_handle_t	192
18.4	Macro Definition Documentation	193
18.4.1	FSL_EDMA_DRIVER_VERSION	193
18.5	Typedef Documentation	193
18.5.1	edma_callback	193
18.6	Enumeration Type Documentation	194

Section No.	Title	Page No.
18.6.1	edma_transfer_size_t	194
18.6.2	edma_modulo_t	194
18.6.3	edma_bandwidth_t	195
18.6.4	edma_channel_link_type_t	195
18.6.5	anonymous enum	195
18.6.6	anonymous enum	195
18.6.7	edma_interrupt_enable_t	196
18.6.8	edma_transfer_type_t	196
18.6.9	anonymous enum	196
18.7	Function Documentation	196
18.7.1	EDMA_Init	196
18.7.2	EDMA_Deinit	197
18.7.3	EDMA_InstallTCD	197
18.7.4	EDMA_GetDefaultConfig	197
18.7.5	EDMA_EnableContinuousChannelLinkMode	198
18.7.6	EDMA_EnableMinorLoopMapping	198
18.7.7	EDMA_ResetChannel	198
18.7.8	EDMA_SetTransferConfig	199
18.7.9	EDMA_SetMinorOffsetConfig	199
18.7.10	EDMA_SetChannelPreemptionConfig	200
18.7.11	EDMA_SetChannelLink	200
18.7.12	EDMA_SetBandWidth	201
18.7.13	EDMA_SetModulo	201
18.7.14	EDMA_EnableAsyncRequest	202
18.7.15	EDMA_EnableAutoStopRequest	202
18.7.16	EDMA_EnableChannelInterrupts	202
18.7.17	EDMA_DisableChannelInterrupts	202
18.7.18	EDMA_SetMajorOffsetConfig	203
18.7.19	EDMA_TcdReset	203
18.7.20	EDMA_TcdSetTransferConfig	203
18.7.21	EDMA_TcdSetMinorOffsetConfig	204
18.7.22	EDMA_TcdSetChannelLink	204
18.7.23	EDMA_TcdSetBandWidth	205
18.7.24	EDMA_TcdSetModulo	205
18.7.25	EDMA_TcdEnableAutoStopRequest	206
18.7.26	EDMA_TcdEnableInterrupts	207
18.7.27	EDMA_TcdDisableInterrupts	207
18.7.28	EDMA_TcdSetMajorOffsetConfig	207
18.7.29	EDMA_EnableChannelRequest	207
18.7.30	EDMA_DisableChannelRequest	208
18.7.31	EDMA_TriggerChannelStart	208
18.7.32	EDMA_GetRemainingMajorLoopCount	208
18.7.33	EDMA_GetErrorStatusFlags	209
18.7.34	EDMA_GetChannelStatusFlags	209

Section No.	Title	Page No.
18.7.35	EDMA_ClearChannelStatusFlags	209
18.7.36	EDMA_CreateHandle	210
18.7.37	EDMA_InstallTCDDMemory	210
18.7.38	EDMA_SetCallback	210
18.7.39	EDMA_PrepareTransferConfig	211
18.7.40	EDMA_PrepareTransfer	211
18.7.41	EDMA_SubmitTransfer	212
18.7.42	EDMA_StartTransfer	213
18.7.43	EDMA_StopTransfer	214
18.7.44	EDMA_AbortTransfer	214
18.7.45	EDMA_GetUnusedTCDDNumber	214
18.7.46	EDMA_GetNextTCDDAddress	214
18.7.47	EDMA_HandleIRQ	215
Chapter 19 ENC: Quadrature Encoder/Decoder		
19.1	Overview	216
19.2	Function groups	216
19.2.1	Initialization and De-initialization	216
19.2.2	Status	216
19.2.3	Interrupts	216
19.2.4	Value Operation	216
19.3	Typical use case	216
19.3.1	Polling Configuration	216
19.4	Data Structure Documentation	219
19.4.1	struct enc_config_t	219
19.4.2	struct enc_self_test_config_t	221
19.5	Enumeration Type Documentation	221
19.5.1	_enc_interrupt_enable	221
19.5.2	_enc_status_flags	222
19.5.3	_enc_signal_status_flags	222
19.5.4	enc_home_trigger_mode_t	222
19.5.5	enc_index_trigger_mode_t	223
19.5.6	enc_decoder_work_mode_t	223
19.5.7	enc_position_match_mode_t	223
19.5.8	enc_revolution_count_condition_t	223
19.5.9	enc_self_test_direction_t	224
19.6	Function Documentation	224
19.6.1	ENC_Init	224
19.6.2	ENC_Deinit	224

Section No.	Title	Page No.
19.6.3	ENC_GetDefaultConfig	224
19.6.4	ENC_DoSoftwareLoadInitialPositionValue	225
19.6.5	ENC_SetSelfTestConfig	225
19.6.6	ENC_EnableWatchdog	225
19.6.7	ENC_SetInitialPositionValue	226
19.6.8	ENC_GetStatusFlags	226
19.6.9	ENC_ClearStatusFlags	226
19.6.10	ENC_GetSignalStatusFlags	226
19.6.11	ENC_EnableInterrupts	227
19.6.12	ENC_DisableInterrupts	227
19.6.13	ENC_GetEnabledInterrupts	227
19.6.14	ENC_GetPositionValue	227
19.6.15	ENC_GetHoldPositionValue	228
19.6.16	ENC_GetPositionDifferenceValue	228
19.6.17	ENC_GetHoldPositionDifferenceValue	228
19.6.18	ENC_GetRevolutionValue	229
19.6.19	ENC_GetHoldRevolutionValue	229

Chapter 20 ENET: Ethernet MAC Driver

20.1	Overview	230
20.2	Operations of Ethernet MAC Driver	230
20.2.1	MII interface Operation	230
20.2.2	MAC address filter	230
20.2.3	Other Basic control Operations	230
20.2.4	Transactional Operation	230
20.2.5	PTP IEEE 1588 Feature Operation	231
20.3	Typical use case	231
20.3.1	ENET Initialization, receive, and transmit operations	231
20.4	Data Structure Documentation	240
20.4.1	struct enet_rx_bd_struct_t	240
20.4.2	struct enet_tx_bd_struct_t	240
20.4.3	struct enet_data_error_stats_t	241
20.4.4	struct enet_rx_frame_error_t	241
20.4.5	struct enet_transfer_stats_t	242
20.4.6	struct enet_frame_info_t	243
20.4.7	struct enet_tx_dirty_ring_t	243
20.4.8	struct enet_buffer_config_t	244
20.4.9	struct enet_intcoalesce_config_t	245
20.4.10	struct enet_avb_config_t	246
20.4.11	struct enet_config_t	246
20.4.12	struct enet_tx_bd_ring_t	249

Section No.	Title	Page No.
20.4.13	struct enet_rx_bd_ring_t	249
20.4.14	struct _enet_handle	250
20.5	Macro Definition Documentation	252
20.5.1	FSL_ENET_DRIVER_VERSION	252
20.5.2	ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK	252
20.5.3	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK	252
20.5.4	ENET_BUFFDESCRIPTOR_RX_WRAP_MASK	252
20.5.5	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask	252
20.5.6	ENET_BUFFDESCRIPTOR_RX_LAST_MASK	252
20.5.7	ENET_BUFFDESCRIPTOR_RX_MISS_MASK	252
20.5.8	ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK	252
20.5.9	ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK	252
20.5.10	ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK	252
20.5.11	ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK	252
20.5.12	ENET_BUFFDESCRIPTOR_RX_CRC_MASK	252
20.5.13	ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK	252
20.5.14	ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK	252
20.5.15	ENET_BUFFDESCRIPTOR_TX_READY_MASK	252
20.5.16	ENET_BUFFDESCRIPTOR_TX_SOFTOWENER1_MASK	252
20.5.17	ENET_BUFFDESCRIPTOR_TX_WRAP_MASK	252
20.5.18	ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_MASK	252
20.5.19	ENET_BUFFDESCRIPTOR_TX_LAST_MASK	252
20.5.20	ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK	252
20.5.21	ENET_BUFFDESCRIPTOR_RX_ERR_MASK	252
20.5.22	ENET_FRAME_MAX_FRAMELEN	253
20.5.23	ENET_FRAME_VLAN_TAGLEN	253
20.5.24	ENET_FRAME_CRC_LEN	253
20.5.25	ENET_FIFO_MIN_RX_FULL	253
20.5.26	ENET_RX_MIN_BUFFERSIZE	253
20.5.27	ENET_PHY_MAXADDRESS	253
20.5.28	ENET_TX_INTERRUPT	253
20.5.29	ENET_RX_INTERRUPT	253
20.5.30	ENET_TS_INTERRUPT	254
20.5.31	ENET_ERR_INTERRUPT	254
20.6	Typedef Documentation	254
20.6.1	enet_rx_alloc_callback_t	254
20.6.2	enet_rx_free_callback_t	254
20.6.3	enet_callback_t	254
20.6.4	enet_isr_ring_t	254
20.7	Enumeration Type Documentation	254
20.7.1	anonymous enum	254
20.7.2	enet_mii_mode_t	255

Section No.	Title	Page No.
20.7.3	enet_mii_speed_t	255
20.7.4	enet_mii_duplex_t	255
20.7.5	enet_mii_write_t	255
20.7.6	enet_mii_read_t	255
20.7.7	enet_mii_extend_opcode	256
20.7.8	enet_special_control_flag_t	256
20.7.9	enet_interrupt_enable_t	256
20.7.10	enet_event_t	257
20.7.11	enet_idle_slope_t	257
20.7.12	enet_tx_accelerator_t	258
20.7.13	enet_rx_accelerator_t	258
20.8	Function Documentation	258
20.8.1	ENET_GetInstance	258
20.8.2	ENET_GetDefaultConfig	259
20.8.3	ENET_Up	259
20.8.4	ENET_Init	260
20.8.5	ENET_Down	261
20.8.6	ENET_Deinit	261
20.8.7	ENET_Reset	262
20.8.8	ENET_SetMII	263
20.8.9	ENET_SetSMI	263
20.8.10	ENET_GetSMI	263
20.8.11	ENET_ReadSMIData	264
20.8.12	ENET_StartSMIWrite	264
20.8.13	ENET_StartSMIRead	264
20.8.14	ENET_MDIOWrite	265
20.8.15	ENET_MDIORead	265
20.8.16	ENET_StartExtC45SMIWriteReg	266
20.8.17	ENET_StartExtC45SMIWriteData	266
20.8.18	ENET_StartExtC45SMIReadData	266
20.8.19	ENET_MDIOC45Write	267
20.8.20	ENET_MDIOC45Read	267
20.8.21	ENET_SetRGMIIIClockDelay	268
20.8.22	ENET_SetMacAddr	269
20.8.23	ENET_GetMacAddr	269
20.8.24	ENET_AddMulticastGroup	269
20.8.25	ENET_LeaveMulticastGroup	269
20.8.26	ENET_ActiveRead	270
20.8.27	ENET_EnableSleepMode	270
20.8.28	ENET_GetAccelFunction	270
20.8.29	ENET_EnableInterrupts	271
20.8.30	ENET_DisableInterrupts	271
20.8.31	ENET_GetInterruptStatus	271
20.8.32	ENET_ClearInterruptStatus	272

Section No.	Title	Page No.
20.8.33	ENET_SetRxISRHandler	272
20.8.34	ENET_SetTxISRHandler	272
20.8.35	ENET_SetErrISRHandler	273
20.8.36	ENET_GetRxErrBeforeReadFrame	273
20.8.37	ENET_GetStatistics	273
20.8.38	ENET_GetRxFrameSize	274
20.8.39	ENET_ReadFrame	274
20.8.40	ENET_SendFrame	275
20.8.41	ENET_SetTxReclaim	276
20.8.42	ENET_ReclaimTxDescriptor	277
20.8.43	ENET_GetRxFrame	277
20.8.44	ENET_StartTxFrame	278
20.8.45	ENET_TransmitIRQHandler	279
20.8.46	ENET_ReceiveIRQHandler	279
20.8.47	ENET_CommonFrame1IRQHandler	279
20.8.48	ENET_CommonFrame2IRQHandler	279
20.8.49	ENET_ErrorIRQHandler	280
20.8.50	ENET_Ptp1588IRQHandler	280
20.8.51	ENET_CommonFrame0IRQHandler	280
20.9	Variable Documentation	280
20.9.1	s_enetClock	280
20.10	ENET CMSIS Driver	281
20.10.1	Typical use case	281
 Chapter 21 EWM: External Watchdog Monitor Driver		
21.1	Overview	283
21.2	Typical use case	283
21.3	Data Structure Documentation	284
21.3.1	struct ewm_config_t	284
21.4	Macro Definition Documentation	284
21.4.1	FSL_EWM_DRIVER_VERSION	284
21.5	Enumeration Type Documentation	284
21.5.1	_ewm_interrupt_enable_t	284
21.5.2	_ewm_status_flags_t	284
21.6	Function Documentation	285
21.6.1	EWM_Init	285
21.6.2	EWM_Deinit	285
21.6.3	EWM_GetDefaultConfig	285

Section No.	Title	Page No.
21.6.4	EWM_EnableInterrupts	286
21.6.5	EWM_DisableInterrupts	286
21.6.6	EWM_GetStatusFlags	286
21.6.7	EWM_Refresh	287
Chapter 22 FlexCAN: Flex Controller Area Network Driver		
22.1	Overview	288
22.2	FlexCAN Driver	289
22.2.1	Overview	289
22.2.2	Typical use case	289
22.2.3	Data Structure Documentation	297
22.2.4	Macro Definition Documentation	304
22.2.5	Enumeration Type Documentation	309
22.2.6	Function Documentation	314
Chapter 23 FlexIO: FlexIO Driver		
23.1	Overview	332
23.2	FlexIO Driver	333
23.2.1	Overview	333
23.2.2	Data Structure Documentation	337
23.2.3	Macro Definition Documentation	340
23.2.4	Typedef Documentation	340
23.2.5	Enumeration Type Documentation	340
23.2.6	Function Documentation	345
23.2.7	Variable Documentation	354
23.3	FlexIO Camera Driver	355
23.3.1	Overview	355
23.3.2	Typical use case	355
23.3.3	Data Structure Documentation	358
23.3.4	Macro Definition Documentation	359
23.3.5	Enumeration Type Documentation	359
23.3.6	Function Documentation	360
23.3.7	FlexIO eDMA Camera Driver	364
23.4	FlexIO I2C Master Driver	368
23.4.1	Overview	368
23.4.2	Typical use case	368
23.4.3	Data Structure Documentation	372
23.4.4	Macro Definition Documentation	375
23.4.5	Typedef Documentation	375

Section No.	Title	Page No.
23.4.6	Enumeration Type Documentation	375
23.4.7	Function Documentation	376
23.5	FlexIO I2S Driver	386
23.5.1	Overview	386
23.5.2	Typical use case	386
23.5.3	Data Structure Documentation	391
23.5.4	Macro Definition Documentation	394
23.5.5	Enumeration Type Documentation	394
23.5.6	Function Documentation	395
23.5.7	FlexIO eDMA I2S Driver	407
23.6	FlexIO MCU Interface LCD Driver	414
23.6.1	Overview	414
23.6.2	Typical use case	414
23.6.3	Data Structure Documentation	420
23.6.4	Macro Definition Documentation	424
23.6.5	Typedef Documentation	424
23.6.6	Enumeration Type Documentation	425
23.6.7	Function Documentation	426
23.6.8	FlexIO eDMA MCU Interface LCD Driver	438
23.7	FlexIO SPI Driver	444
23.7.1	Overview	444
23.7.2	Typical use case	444
23.7.3	Data Structure Documentation	451
23.7.4	Macro Definition Documentation	455
23.7.5	Typedef Documentation	455
23.7.6	Enumeration Type Documentation	455
23.7.7	Function Documentation	457
23.7.8	FlexIO eDMA SPI Driver	472
23.8	FlexIO UART Driver	480
23.8.1	Overview	480
23.8.2	Typical use case	480
23.8.3	Data Structure Documentation	489
23.8.4	Macro Definition Documentation	492
23.8.5	Typedef Documentation	492
23.8.6	Enumeration Type Documentation	492
23.8.7	Function Documentation	493
23.8.8	FlexIO eDMA UART Driver	504

Chapter 24 FLEXRAM: on-chip RAM manager

24.1	Overview	510
-------------	-----------------------	------------

Section No.	Title	Page No.
24.2	Data Structure Documentation	512
24.2.1	struct flexram_allocate_ram_t	512
24.3	Macro Definition Documentation	512
24.3.1	FSL_SOC_FLEXRAM_ALLOCATE_DRIVER_VERSION	512
24.3.2	FSL_FLEXRAM_DRIVER_VERSION	512
24.3.3	FLEXRAM_ECC_ERROR_DETAILED_INFO	512
24.4	Enumeration Type Documentation	512
24.4.1	anonymous enum	512
24.4.2	flexram_bank_allocate_src_t	513
24.4.3	anonymous enum	513
24.4.4	anonymous enum	513
24.4.5	flexram_tcm_access_mode_t	513
24.4.6	anonymous enum	513
24.5	Function Documentation	514
24.5.1	FLEXRAM_AllocateRam	514
24.5.2	FLEXRAM_SetAllocateRamSrc	514
24.5.3	FLEXRAM_Init	514
24.5.4	FLEXRAM_GetInterruptStatus	514
24.5.5	FLEXRAM_ClearInterruptStatus	515
24.5.6	FLEXRAM_EnableInterruptStatus	515
24.5.7	FLEXRAM_DisableInterruptStatus	515
24.5.8	FLEXRAM_EnableInterruptSignal	515
24.5.9	FLEXRAM_DisableInterruptSignal	516
24.5.10	FLEXRAM_SetTCMReadAccessMode	516
24.5.11	FLEXRAM_SetTCMWriteAccessMode	516
24.5.12	FLEXRAM_EnableForceRamClockOn	516
 Chapter 25 FLEXSPI: Flexible Serial Peripheral Interface Driver		
25.1	Overview	518
25.2	Data Structure Documentation	524
25.2.1	struct flexspi_config_t	524
25.2.2	struct flexspi_device_config_t	527
25.2.3	struct flexspi_transfer_t	528
25.2.4	struct _flexspi_handle	529
25.3	Macro Definition Documentation	530
25.3.1	FSL_FLEXSPI_DRIVER_VERSION	530
25.3.2	FLEXSPI_LUT_SEQ	530
25.4	Typedef Documentation	530
25.4.1	flexspi_transfer_callback_t	530

Section No.	Title	Page No.
25.5	Enumeration Type Documentation	530
25.5.1	anonymous enum	530
25.5.2	anonymous enum	530
25.5.3	flexspi_pad_t	531
25.5.4	flexspi_flags_t	531
25.5.5	flexspi_read_sample_clock_t	532
25.5.6	flexspi_cs_interval_cycle_unit_t	532
25.5.7	flexspi_ahb_write_wait_unit_t	533
25.5.8	flexspi_ip_error_code_t	533
25.5.9	flexspi_ahb_error_code_t	533
25.5.10	flexspi_port_t	534
25.5.11	flexspi_arb_command_source_t	534
25.5.12	flexspi_command_type_t	534
25.6	Function Documentation	534
25.6.1	FLEXSPI_GetInstance	534
25.6.2	FLEXSPI_CheckAndClearError	534
25.6.3	FLEXSPI_Init	534
25.6.4	FLEXSPI_GetDefaultConfig	535
25.6.5	FLEXSPI_Deinit	535
25.6.6	FLEXSPI_UpdateDIIValue	535
25.6.7	FLEXSPI_SetFlashConfig	535
25.6.8	FLEXSPI_SoftwareReset	536
25.6.9	FLEXSPI_Enable	536
25.6.10	FLEXSPI_EnableInterrupts	536
25.6.11	FLEXSPI_DisableInterrupts	536
25.6.12	FLEXSPI_EnableTxDMA	537
25.6.13	FLEXSPI_EnableRxDMA	537
25.6.14	FLEXSPI_GetTxFifoAddress	537
25.6.15	FLEXSPI_GetRxFifoAddress	537
25.6.16	FLEXSPI_ResetFifos	538
25.6.17	FLEXSPI_GetFifoCounts	538
25.6.18	FLEXSPI_GetInterruptStatusFlags	538
25.6.19	FLEXSPI_ClearInterruptStatusFlags	539
25.6.20	FLEXSPI_GetDataLearningPhase	539
25.6.21	FLEXSPI_GetArbitratorCommandSource	539
25.6.22	FLEXSPI_GetIPCommandErrorCode	540
25.6.23	FLEXSPI_GetAHBCommandErrorCode	540
25.6.24	FLEXSPI_GetBusIdleStatus	540
25.6.25	FLEXSPI_UpdateRxSampleClock	541
25.6.26	FLEXSPI_EnableIPParallelMode	541
25.6.27	FLEXSPI_EnableAHBParallelMode	541
25.6.28	FLEXSPI_UpdateLUT	541
25.6.29	FLEXSPI_WriteData	542
25.6.30	FLEXSPI_ReadData	542

Section No.	Title	Page No.
25.6.31	FLEXSPI_WriteBlocking	542
25.6.32	FLEXSPI_ReadBlocking	543
25.6.33	FLEXSPI_TransferBlocking	544
25.6.34	FLEXSPI_TransferCreateHandle	544
25.6.35	FLEXSPI_TransferNonBlocking	545
25.6.36	FLEXSPI_TransferGetCount	545
25.6.37	FLEXSPI_TransferAbort	546
25.6.38	FLEXSPI_TransferHandleIRQ	546
25.7	FLEXSPI eDMA Driver	547
25.7.1	Overview	547
25.7.2	Data Structure Documentation	548
25.7.3	Macro Definition Documentation	549
25.7.4	Enumeration Type Documentation	549
25.7.5	Function Documentation	549
Chapter 26 GPC: General Power Controller Driver		
26.1	Overview	552
26.2	Macro Definition Documentation	552
26.2.1	FSL_GPC_DRIVER_VERSION	552
26.3	Function Documentation	552
26.3.1	GPC_EnableIRQ	552
26.3.2	GPC_DisableIRQ	553
26.3.3	GPC_GetIRQStatusFlag	553
26.3.4	GPC_RequestMEGAPowerOn	553
Chapter 27 GPT: General Purpose Timer		
27.1	Overview	554
27.2	Function groups	554
27.2.1	Initialization and deinitialization	554
27.3	Typical use case	554
27.3.1	GPT interrupt example	554
27.4	Data Structure Documentation	557
27.4.1	struct gpt_config_t	557
27.5	Enumeration Type Documentation	558
27.5.1	gpt_clock_source_t	558
27.5.2	gpt_input_capture_channel_t	558
27.5.3	gpt_input_operation_mode_t	559

Section No.	Title	Page No.
27.5.4	<code>gpt_output_compare_channel_t</code>	559
27.5.5	<code>gpt_output_operation_mode_t</code>	559
27.5.6	<code>gpt_interrupt_enable_t</code>	559
27.5.7	<code>gpt_status_flag_t</code>	560
27.6	Function Documentation	560
27.6.1	<code>GPT_Init</code>	560
27.6.2	<code>GPT_Deinit</code>	560
27.6.3	<code>GPT_GetDefaultConfig</code>	560
27.6.4	<code>GPT_SoftwareReset</code>	561
27.6.5	<code>GPT_SetClockSource</code>	561
27.6.6	<code>GPT_GetClockSource</code>	561
27.6.7	<code>GPT_SetClockDivider</code>	561
27.6.8	<code>GPT_GetClockDivider</code>	562
27.6.9	<code>GPT_SetOscClockDivider</code>	562
27.6.10	<code>GPT_GetOscClockDivider</code>	562
27.6.11	<code>GPT_StartTimer</code>	562
27.6.12	<code>GPT_StopTimer</code>	563
27.6.13	<code>GPT_GetCurrentTimerCount</code>	563
27.6.14	<code>GPT_SetInputOperationMode</code>	563
27.6.15	<code>GPT_GetInputOperationMode</code>	563
27.6.16	<code>GPT_GetInputCaptureValue</code>	564
27.6.17	<code>GPT_SetOutputOperationMode</code>	564
27.6.18	<code>GPT_GetOutputOperationMode</code>	565
27.6.19	<code>GPT_SetOutputCompareValue</code>	565
27.6.20	<code>GPT_GetOutputCompareValue</code>	565
27.6.21	<code>GPT_ForceOutput</code>	566
27.6.22	<code>GPT_EnableInterrupts</code>	566
27.6.23	<code>GPT_DisableInterrupts</code>	566
27.6.24	<code>GPT_GetEnabledInterrupts</code>	566
27.6.25	<code>GPT_GetStatusFlags</code>	567
27.6.26	<code>GPT_ClearStatusFlags</code>	567
Chapter 28	GPIO: General-Purpose Input/Output Driver	
28.1	Overview	568
28.2	Typical use case	568
28.2.1	Input Operation	568
28.3	Data Structure Documentation	570
28.3.1	<code>struct gpio_pin_config_t</code>	570
28.4	Macro Definition Documentation	570
28.4.1	<code>FSL_GPIO_DRIVER_VERSION</code>	570

Section No.	Title	Page No.
28.5	Enumeration Type Documentation	570
28.5.1	gpio_pin_direction_t	570
28.5.2	gpio_interrupt_mode_t	570
28.6	Function Documentation	571
28.6.1	GPIO_PinInit	571
28.6.2	GPIO_PinWrite	572
28.6.3	GPIO_WritePinOutput	572
28.6.4	GPIO_PortSet	572
28.6.5	GPIO_SetPinsOutput	572
28.6.6	GPIO_PortClear	573
28.6.7	GPIO_ClearPinsOutput	574
28.6.8	GPIO_PortToggle	574
28.6.9	GPIO_PinRead	574
28.6.10	GPIO_ReadPinInput	574
28.6.11	GPIO_PinReadPadStatus	575
28.6.12	GPIO_ReadPadStatus	576
28.6.13	GPIO_PinSetInterruptConfig	576
28.6.14	GPIO_SetPinInterruptConfig	576
28.6.15	GPIO_PortEnableInterrupts	576
28.6.16	GPIO_EnableInterrupts	577
28.6.17	GPIO_PortDisableInterrupts	577
28.6.18	GPIO_DisableInterrupts	577
28.6.19	GPIO_PortGetInterruptFlags	577
28.6.20	GPIO_GetPinsInterruptFlags	578
28.6.21	GPIO_PortClearInterruptFlags	578
28.6.22	GPIO_ClearPinsInterruptFlags	578
 Chapter 29 KPP: KeyPad Port Driver		
29.1	Overview	580
29.2	Typical use case	580
29.3	Data Structure Documentation	581
29.3.1	struct kpp_config_t	581
29.4	Macro Definition Documentation	581
29.4.1	FSL_KPP_DRIVER_VERSION	581
29.5	Enumeration Type Documentation	582
29.5.1	kpp_interrupt_enable_t	582
29.5.2	kpp_sync_operation_t	582
29.6	Function Documentation	582
29.6.1	KPP_Init	582

Section No.	Title	Page No.
29.6.2	KPP_Deinit	582
29.6.3	KPP_EnableInterrupts	582
29.6.4	KPP_DisableInterrupts	583
29.6.5	KPP_GetStatusFlag	583
29.6.6	KPP_ClearStatusFlag	583
29.6.7	KPP_SetSynchronizeChain	584
29.6.8	KPP_keyPressScanning	585
Chapter 30 LPI2C: Low Power Inter-Integrated Circuit Driver		
30.1	Overview	586
30.2	Macro Definition Documentation	587
30.2.1	FSL_LPI2C_DRIVER_VERSION	587
30.2.2	I2C_RETRY_TIMES	587
30.3	Enumeration Type Documentation	587
30.3.1	anonymous enum	587
30.4	LPI2C Master Driver	588
30.4.1	Overview	588
30.4.2	Data Structure Documentation	591
30.4.3	Typedef Documentation	595
30.4.4	Enumeration Type Documentation	596
30.4.5	Function Documentation	598
30.5	LPI2C Slave Driver	612
30.5.1	Overview	612
30.5.2	Data Structure Documentation	614
30.5.3	Typedef Documentation	618
30.5.4	Enumeration Type Documentation	619
30.5.5	Function Documentation	620
30.6	LPI2C Master DMA Driver	629
30.6.1	Overview	629
30.6.2	Data Structure Documentation	629
30.6.3	Typedef Documentation	631
30.6.4	Function Documentation	632
30.7	LPI2C FreeRTOS Driver	635
30.7.1	Overview	635
30.7.2	Macro Definition Documentation	635
30.7.3	Function Documentation	635
30.8	LPI2C CMSIS Driver	638
30.8.1	LPI2C CMSIS Driver	638

Section No.	Title	Page No.
Chapter 31 LPSPI: Low Power Serial Peripheral Interface		
31.1	Overview	640
31.2	LPSPI Peripheral driver	641
31.2.1	Overview	641
31.2.2	Function groups	641
31.2.3	Typical use case	641
31.2.4	Data Structure Documentation	648
31.2.5	Macro Definition Documentation	655
31.2.6	Typedef Documentation	656
31.2.7	Enumeration Type Documentation	657
31.2.8	Function Documentation	662
31.2.9	Variable Documentation	678
31.3	LPSPI eDMA Driver	679
31.3.1	Overview	679
31.3.2	Data Structure Documentation	680
31.3.3	Macro Definition Documentation	685
31.3.4	Typedef Documentation	685
31.3.5	Function Documentation	686
31.4	LPSPI FreeRTOS Driver	693
31.4.1	Overview	693
31.4.2	Macro Definition Documentation	693
31.4.3	Function Documentation	693
31.5	LPSPI CMSIS Driver	696
31.5.1	Function groups	696
31.5.2	Typical use case	697
Chapter 32 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver		
32.1	Overview	698
32.2	LPUART Driver	699
32.2.1	Overview	699
32.2.2	Typical use case	699
32.2.3	Data Structure Documentation	703
32.2.4	Macro Definition Documentation	707
32.2.5	Typedef Documentation	707
32.2.6	Enumeration Type Documentation	707
32.2.7	Function Documentation	710
32.3	LPUART eDMA Driver	724
32.3.1	Overview	724

Section No.	Title	Page No.
32.3.2	Data Structure Documentation	725
32.3.3	Macro Definition Documentation	726
32.3.4	Typedef Documentation	726
32.3.5	Function Documentation	726
32.4	LPUART FreeRTOS Driver	731
32.4.1	Overview	731
32.4.2	Data Structure Documentation	731
32.4.3	Macro Definition Documentation	732
32.4.4	Function Documentation	732
32.5	LPUART CMSIS Driver	735
32.5.1	Function groups	735
 Chapter 33 OCOTP: On Chip One-Time Programmable controller.		
33.1	Overview	737
33.2	OCOTP function group	737
33.2.1	Initialization and de-initialization	737
33.2.2	Read and Write operation	737
33.3	OCOTP example	737
33.4	Macro Definition Documentation	738
33.4.1	FSL_OCOTP_DRIVER_VERSION	738
33.5	Enumeration Type Documentation	739
33.5.1	anonymous enum	739
33.6	Function Documentation	739
33.6.1	OCOTP_Init	739
33.6.2	OCOTP_Deinit	739
33.6.3	OCOTP_CheckBusyStatus	739
33.6.4	OCOTP_CheckErrorStatus	740
33.6.5	OCOTP_ClearErrorStatus	740
33.6.6	OCOTP_ReloadShadowRegister	740
33.6.7	OCOTP_ReadFuseShadowRegister	741
33.6.8	OCOTP_ReadFuseShadowRegisterExt	741
33.6.9	OCOTP_WriteFuseShadowRegister	742
33.6.10	OCOTP_WriteFuseShadowRegisterWithLock	742
33.6.11	OCOTP_GetVersion	743
 Chapter 34 PIT: Periodic Interrupt Timer		
34.1	Overview	744

Section No.	Title	Page No.
34.2	Function groups	744
34.2.1	Initialization and deinitialization	744
34.2.2	Timer period Operations	744
34.2.3	Start and Stop timer operations	744
34.2.4	Status	745
34.2.5	Interrupt	745
34.3	Typical use case	745
34.3.1	PIT tick example	745
34.4	Data Structure Documentation	746
34.4.1	struct pit_config_t	746
34.5	Enumeration Type Documentation	746
34.5.1	pit_chnl_t	746
34.5.2	pit_interrupt_enable_t	747
34.5.3	pit_status_flags_t	747
34.6	Function Documentation	747
34.6.1	PIT_Init	747
34.6.2	PIT_Deinit	747
34.6.3	PIT_GetDefaultConfig	748
34.6.4	PIT_EnableInterrupts	748
34.6.5	PIT_DisableInterrupts	748
34.6.6	PIT_GetEnabledInterrupts	749
34.6.7	PIT_GetStatusFlags	749
34.6.8	PIT_ClearStatusFlags	749
34.6.9	PIT_SetTimerPeriod	750
34.6.10	PIT_GetCurrentTimerCount	750
34.6.11	PIT_StartTimer	751
34.6.12	PIT_StopTimer	751
 Chapter 35 PMU: Power Management Unit		
35.1	Overview	752
35.2	Macro Definition Documentation	754
35.2.1	FSL_PMU_DRIVER_VERSION	754
35.3	Enumeration Type Documentation	754
35.3.1	anonymous enum	754
35.3.2	pmu_1p1_weak_reference_source_t	755
35.3.3	pmu_3p0_vbus_voltage_source_t	755
35.3.4	pmu_core_reg_voltage_ramp_rate_t	755
35.3.5	pmu_power_bandgap_t	755

Section No.	Title	Page No.
35.4	Function Documentation	755
35.4.1	PMU_GetStatusFlags	755
35.4.2	PMU_1P1SetWeakReferenceSource	756
35.4.3	PMU_1P1EnableWeakRegulator	756
35.4.4	PMU_1P1SetRegulatorOutputVoltage	756
35.4.5	PMU_1P1SetBrownoutOffsetVoltage	757
35.4.6	PMU_1P1EnablePullDown	757
35.4.7	PMU_1P1EnableCurrentLimit	757
35.4.8	PMU_1P1EnableBrownout	757
35.4.9	PMU_1P1EnableOutput	758
35.4.10	PMU_3P0SetRegulatorOutputVoltage	758
35.4.11	PMU_3P0SetVBusVoltageSource	758
35.4.12	PMU_3P0SetBrownoutOffsetVoltage	759
35.4.13	PMU_3P0EnableCurrentLimit	759
35.4.14	PMU_3P0EnableBrownout	759
35.4.15	PMU_3P0EnableOutput	759
35.4.16	PMU_2P5EnableWeakRegulator	760
35.4.17	PMU_2P5SetRegulatorOutputVoltage	760
35.4.18	PMU_2P5SetBrownoutOffsetVoltage	760
35.4.19	PMU_2P5EnablePullDown	761
35.4.20	PMU_2P1EnablePullDown	761
35.4.21	PMU_2P5EnableCurrentLimit	761
35.4.22	PMU_2P5nableBrownout	761
35.4.23	PMU_2P5EnableOutput	762
35.4.24	PMU_CoreEnableIncreaseGateDrive	763
35.4.25	PMU_CoreSetRegulatorVoltageRampRate	763
35.4.26	PMU_CoreSetSOCDomainVoltage	763
35.4.27	PMU_CoreSetARMCoreDomainVoltage	764

Chapter 36 PWM: Pulse Width Modulator

36.1	Overview	765
36.2	PWM: Pulse Width Modulator	765
36.2.1	Initialization and deinitialization	765
36.2.2	PWM Operations	765
36.2.3	Input capture operations	765
36.2.4	Fault operation	765
36.2.5	PWM Start and Stop operations	766
36.2.6	Status	766
36.2.7	Interrupt	766
36.3	Register Update	766
36.4	Typical use case	766

Section No.	Title	Page No.
36.4.1	PWM output	766
36.5	Data Structure Documentation	775
36.5.1	struct pwm_signal_param_t	775
36.5.2	struct pwm_config_t	775
36.5.3	struct pwm_fault_input_filter_param_t	776
36.5.4	struct pwm_fault_param_t	776
36.5.5	struct pwm_input_capture_param_t	777
36.6	Macro Definition Documentation	777
36.6.1	PWM_SUBMODULE_CHANNEL	777
36.7	Enumeration Type Documentation	777
36.7.1	pwm_submodule_t	777
36.7.2	pwm_value_register_t	777
36.7.3	_pwm_value_register_mask	778
36.7.4	pwm_clock_source_t	778
36.7.5	pwm_clock_prescale_t	778
36.7.6	pwm_force_output_trigger_t	778
36.7.7	pwm_output_state_t	779
36.7.8	pwm_init_source_t	779
36.7.9	pwm_load_frequency_t	779
36.7.10	pwm_fault_input_t	780
36.7.11	pwm_fault_disable_t	780
36.7.12	pwm_input_capture_edge_t	780
36.7.13	pwm_force_signal_t	781
36.7.14	pwm_chnl_pair_operation_t	781
36.7.15	pwm_register_reload_t	781
36.7.16	pwm_fault_recovery_mode_t	781
36.7.17	pwm_interrupt_enable_t	782
36.7.18	pwm_status_flags_t	782
36.7.19	pwm_dma_enable_t	783
36.7.20	pwm_dma_source_select_t	783
36.7.21	pwm_watermark_control_t	783
36.7.22	pwm_mode_t	783
36.7.23	pwm_level_select_t	784
36.7.24	pwm_fault_state_t	784
36.7.25	pwm_reload_source_select_t	784
36.7.26	pwm_fault_clear_t	784
36.7.27	pwm_module_control_t	784
36.8	Function Documentation	785
36.8.1	PWM_Init	785
36.8.2	PWM_Deinit	785
36.8.3	PWM_GetDefaultConfig	785

Section No.	Title	Page No.
36.8.4	PWM_SetupPwm	786
36.8.5	PWM_SetupPwmPhaseShift	786
36.8.6	PWM_UpdatePwmDutycycle	787
36.8.7	PWM_UpdatePwmDutycycleHighAccuracy	787
36.8.8	PWM_SetupInputCapture	788
36.8.9	PWM_SetupFaultInputFilter	788
36.8.10	PWM_SetupFaults	788
36.8.11	PWM_FaultDefaultConfig	789
36.8.12	PWM_SetupForceSignal	789
36.8.13	PWM_EnableInterrupts	789
36.8.14	PWM_DisableInterrupts	790
36.8.15	PWM_GetEnabledInterrupts	790
36.8.16	PWM_DMAFIFOWatermarkControl	790
36.8.17	PWM_DMACaptureSourceSelect	791
36.8.18	PWM_EnableDMACapture	791
36.8.19	PWM_EnableDMAWrite	791
36.8.20	PWM_GetStatusFlags	792
36.8.21	PWM_ClearStatusFlags	792
36.8.22	PWM_StartTimer	792
36.8.23	PWM_StopTimer	793
36.8.24	PWM_SetVALxValue	793
36.8.25	PWM_GetVALxValue	793
36.8.26	PWM_OutputTriggerEnable	794
36.8.27	PWM_ActivateOutputTrigger	794
36.8.28	PWM_DeactivateOutputTrigger	795
36.8.29	PWM_SetupSwCtrlOut	796
36.8.30	PWM_SetPwmLdok	796
36.8.31	PWM_SetPwmFaultState	797
36.8.32	PWM_SetupFaultDisableMap	797
36.8.33	PWM_OutputEnable	797
36.8.34	PWM_OutputDisable	798
36.8.35	PWM_GetPwmChannelState	798
36.8.36	PWM_SetOutputToIdle	798
36.8.37	PWM_SetClockMode	799
36.8.38	PWM_SetPwmForceOutputToZero	799
36.8.39	PWM_SetChannelOutput	799

Chapter 37 QTMR: Quad Timer Driver

37.1	Overview	801
37.2	Data Structure Documentation	805
37.2.1	struct qtmr_config_t	805
37.3	Enumeration Type Documentation	805

Section No.	Title	Page No.
37.3.1	qtmr_primary_count_source_t	805
37.3.2	qtmr_input_source_t	806
37.3.3	qtmr_counting_mode_t	806
37.3.4	qtmr_pwm_out_state_t	806
37.3.5	qtmr_output_mode_t	807
37.3.6	qtmr_input_capture_edge_t	807
37.3.7	qtmr_preload_control_t	807
37.3.8	qtmr_debug_action_t	807
37.3.9	qtmr_interrupt_enable_t	808
37.3.10	qtmr_status_flags_t	808
37.3.11	qtmr_channel_selection_t	808
37.3.12	qtmr_dma_enable_t	808
37.4	Function Documentation	808
37.4.1	QTMR_Init	808
37.4.2	QTMR_Deinit	809
37.4.3	QTMR_GetDefaultConfig	809
37.4.4	QTMR_SetupPwm	809
37.4.5	QTMR_SetupInputCapture	810
37.4.6	QTMR_EnableInterrupts	810
37.4.7	QTMR_DisableInterrupts	811
37.4.8	QTMR_GetEnabledInterrupts	811
37.4.9	QTMR_GetStatus	811
37.4.10	QTMR_ClearStatusFlags	812
37.4.11	QTMR_SetTimerPeriod	812
37.4.12	QTMR_SetCompareValue	813
37.4.13	QTMR_SetLoadValue	813
37.4.14	QTMR_GetCurrentTimerCount	813
37.4.15	QTMR_StartTimer	814
37.4.16	QTMR_StopTimer	814
37.4.17	QTMR_EnableDma	814
37.4.18	QTMR_DisableDma	815
37.4.19	QTMR_SetPwmOutputToIdle	815
37.4.20	QTMR_GetPwmOutputStatus	815
37.4.21	QTMR_GetPwmChannelStatus	816
37.4.22	QTMR_SetPwmClockMode	816
Chapter 38 RTWDOG: 32-bit Watchdog Timer		
38.1	Overview	817
38.2	Typical use case	817
38.3	Data Structure Documentation	819
38.3.1	struct rtwdog_work_mode_t	819

Section No.	Title	Page No.
38.3.2	struct <code>rtwdog_config_t</code>	819
38.4	Macro Definition Documentation	820
38.4.1	<code>FSL_RTWD OG_DRIVER_VERSION</code>	820
38.5	Enumeration Type Documentation	820
38.5.1	<code>rtwdog_clock_source_t</code>	820
38.5.2	<code>rtwdog_clock_prescaler_t</code>	820
38.5.3	<code>rtwdog_test_mode_t</code>	820
38.5.4	<code>_rtwdog_interrupt_enable_t</code>	820
38.5.5	<code>_rtwdog_status_flags_t</code>	821
38.6	Function Documentation	821
38.6.1	<code>RTWD OG_GetDefaultConfig</code>	821
38.6.2	<code>RTWD OG_Init</code>	821
38.6.3	<code>RTWD OG_Deinit</code>	822
38.6.4	<code>RTWD OG_Enable</code>	822
38.6.5	<code>RTWD OG_Disable</code>	822
38.6.6	<code>RTWD OG_EnableInterrupts</code>	823
38.6.7	<code>RTWD OG_DisableInterrupts</code>	823
38.6.8	<code>RTWD OG_GetStatusFlags</code>	823
38.6.9	<code>RTWD OG_EnableWindowMode</code>	824
38.6.10	<code>RTWD OG_CountToMesec</code>	824
38.6.11	<code>RTWD OG_ClearStatusFlags</code>	824
38.6.12	<code>RTWD OG_SetTimeoutValue</code>	825
38.6.13	<code>RTWD OG_SetWindowValue</code>	825
38.6.14	<code>RTWD OG_Unlock</code>	825
38.6.15	<code>RTWD OG_Refresh</code>	826
38.6.16	<code>RTWD OG_GetCounterValue</code>	826
 Chapter 39 SAI: Serial Audio Interface		
39.1	Overview	827
39.2	Typical configurations	827
39.3	Typical use case	828
39.3.1	SAI Send/receive using an interrupt method	828
39.3.2	SAI Send/receive using a DMA method	828
39.4	SAI Driver	829
39.4.1	Overview	829
39.4.2	Data Structure Documentation	837
39.4.3	Macro Definition Documentation	841
39.4.4	Enumeration Type Documentation	841
39.4.5	Function Documentation	845

Section No.	Title	Page No.
39.5	SAI EDMA Driver	876
39.5.1	Overview	876
39.5.2	Data Structure Documentation	877
39.5.3	Function Documentation	878
Chapter 40 SEMC: Smart External DRAM Controller Driver		
40.1	Overview	889
40.2	SEMC: Smart External DRAM Controller Driver	889
40.2.1	SEMC Initialization Operation	889
40.2.2	SEMC Interrupt Operation	889
40.2.3	SEMC Memory access Operation	889
40.3	Typical use case	889
40.4	Data Structure Documentation	896
40.4.1	struct semc_sdram_config_t	896
40.4.2	struct semc_nand_timing_config_t	898
40.4.3	struct semc_nand_config_t	900
40.4.4	struct semc_nor_config_t	901
40.4.5	struct semc_sram_config_t	904
40.4.6	struct semc_dbi_config_t	906
40.4.7	struct semc_queuea_weight_struct_t	908
40.4.8	union semc_queuea_weight_t	909
40.4.9	struct semc_queueb_weight_struct_t	909
40.4.10	union semc_queueb_weight_t	910
40.4.11	struct semc_axi_queueweight_t	910
40.4.12	struct semc_config_t	911
40.5	Macro Definition Documentation	911
40.5.1	FSL_SEMC_DRIVER_VERSION	911
40.6	Enumeration Type Documentation	911
40.6.1	anonymous enum	911
40.6.2	semc_mem_type_t	912
40.6.3	semc_waitready_polarity_t	912
40.6.4	semc_sdram_cs_t	912
40.6.5	semc_sram_cs_t	912
40.6.6	semc_nand_access_type_t	913
40.6.7	semc_interrupt_enable_t	913
40.6.8	semc_ipcmd_datasize_t	913
40.6.9	semc_refresh_time_t	913
40.6.10	semc_caslatency_t	913
40.6.11	semc_sdram_column_bit_num_t	914

Section No.	Title	Page No.
40.6.12	sem_sdram_burst_len_t	914
40.6.13	semc_nand_column_bit_num_t	914
40.6.14	sem_nand_burst_len_t	914
40.6.15	semc_norsram_column_bit_num_t	915
40.6.16	sem_norsram_burst_len_t	915
40.6.17	semc_dbi_column_bit_num_t	915
40.6.18	sem_dbi_burst_len_t	916
40.6.19	semc_iomux_pin	916
40.6.20	semc_iomux_nora27_pin	916
40.6.21	smec_port_size_t	917
40.6.22	semc_addr_mode_t	917
40.6.23	semc_dqs_mode_t	917
40.6.24	semc_adv_polarity_t	917
40.6.25	semc_sync_mode_t	917
40.6.26	semc_adv_level_control_t	918
40.6.27	semc_rdy_polarity_t	918
40.6.28	semc_ipcmd_nand_addrmode_t	918
40.6.29	semc_ipcmd_nand_cmdmode_t	918
40.6.30	semc_nand_address_option_t	919
40.6.31	semc_ipcmd_nor_dbi_t	919
40.6.32	semc_ipcmd_sram_t	919
40.6.33	semc_ipcmd_sdram_t	919
40.7	Function Documentation	920
40.7.1	SEMC_GetDefaultConfig	920
40.7.2	SEMC_Init	920
40.7.3	SEMC_Deinit	920
40.7.4	SEMC_ConfigureSDRAM	920
40.7.5	SEMC_ConfigureNAND	921
40.7.6	SEMC_ConfigureNOR	921
40.7.7	SEMC_ConfigureSRAMWithChipSelection	921
40.7.8	SEMC_ConfigureSRAM	922
40.7.9	SEMC_ConfigureDBI	922
40.7.10	SEMC_EnableInterrupts	922
40.7.11	SEMC_DisableInterrupts	923
40.7.12	SEMC_GetStatusFlag	923
40.7.13	SEMC_ClearStatusFlags	923
40.7.14	SEMC_IsInIdle	924
40.7.15	SEMC_SendIPCommand	924
40.7.16	SEMC_BuildNandIPCommand	924
40.7.17	SEMC_IsNandReady	925
40.7.18	SEMC_IPCommandNandWrite	925
40.7.19	SEMC_IPCommandNandRead	925
40.7.20	SEMC_IPCommandNorWrite	926
40.7.21	SEMC_IPCommandNorRead	926

Section No.	Title	Page No.
Chapter 41 SNVS: Secure Non-Volatile Storage		
41.1	Overview	927
41.2	Secure Non-Volatile Storage High-Power	928
41.2.1	Overview	928
41.2.2	Data Structure Documentation	931
41.2.3	Macro Definition Documentation	932
41.2.4	Enumeration Type Documentation	933
41.2.5	Function Documentation	934
41.3	Secure Non-Volatile Storage Low-Power	944
41.3.1	Overview	944
41.3.2	Data Structure Documentation	947
41.3.3	Enumeration Type Documentation	948
41.3.4	Function Documentation	948
Chapter 42 SPDIF: Sony/Philips Digital Interface		
42.1	Overview	956
42.2	Typical use case	956
42.2.1	SPDIF Send/receive using an interrupt method	956
42.2.2	SPDIF Send/receive using a DMA method	956
42.3	Data Structure Documentation	961
42.3.1	struct spdif_config_t	961
42.3.2	struct spdif_transfer_t	962
42.3.3	struct _spdif_handle	962
42.4	Macro Definition Documentation	962
42.4.1	SPDIF_XFER_QUEUE_SIZE	962
42.5	Enumeration Type Documentation	963
42.5.1	anonymous enum	963
42.5.2	spdif_rxfull_select_t	963
42.5.3	spdif_txempty_select_t	963
42.5.4	spdif_uchannel_source_t	964
42.5.5	spdif_gain_select_t	964
42.5.6	spdif_tx_source_t	964
42.5.7	spdif_validity_config_t	964
42.5.8	anonymous enum	964
42.5.9	anonymous enum	965
42.6	Function Documentation	965
42.6.1	SPDIF_Init	965

Section No.	Title	Page No.
42.6.2	SPDIF_GetDefaultConfig	966
42.6.3	SPDIF_Deinit	966
42.6.4	SPDIF_GetInstance	966
42.6.5	SPDIF_TxFIFOReset	966
42.6.6	SPDIF_RxFIFOReset	967
42.6.7	SPDIF_TxEnable	967
42.6.8	SPDIF_RxEnable	967
42.6.9	SPDIF_GetStatusFlag	967
42.6.10	SPDIF_ClearStatusFlags	968
42.6.11	SPDIF_EnableInterrupts	968
42.6.12	SPDIF_DisableInterrupts	968
42.6.13	SPDIF_EnableDMA	969
42.6.14	SPDIF_TxGetLeftDataRegisterAddress	969
42.6.15	SPDIF_TxGetRightDataRegisterAddress	970
42.6.16	SPDIF_RxGetLeftDataRegisterAddress	971
42.6.17	SPDIF_RxGetRightDataRegisterAddress	971
42.6.18	SPDIF_TxSetSampleRate	971
42.6.19	SPDIF_GetRxSampleRate	972
42.6.20	SPDIF_WriteBlocking	972
42.6.21	SPDIF_WriteLeftData	972
42.6.22	SPDIF_WriteRightData	973
42.6.23	SPDIF_WriteChannelStatusHigh	973
42.6.24	SPDIF_WriteChannelStatusLow	973
42.6.25	SPDIF_ReadBlocking	973
42.6.26	SPDIF_ReadLeftData	974
42.6.27	SPDIF_ReadRightData	974
42.6.28	SPDIF_ReadChannelStatusHigh	974
42.6.29	SPDIF_ReadChannelStatusLow	975
42.6.30	SPDIF_ReadQChannel	976
42.6.31	SPDIF_ReadUChannel	976
42.6.32	SPDIF_TransferTxCreateHandle	976
42.6.33	SPDIF_TransferRxCreateHandle	977
42.6.34	SPDIF_TransferSendNonBlocking	977
42.6.35	SPDIF_TransferReceiveNonBlocking	978
42.6.36	SPDIF_TransferGetSendCount	978
42.6.37	SPDIF_TransferGetReceiveCount	979
42.6.38	SPDIF_TransferAbortSend	979
42.6.39	SPDIF_TransferAbortReceive	979
42.6.40	SPDIF_TransferTxHandleIRQ	980
42.6.41	SPDIF_TransferRxHandleIRQ	980
42.7	SPDIF eDMA Driver	981
42.7.1	Overview	981
42.7.2	Data Structure Documentation	982
42.7.3	Function Documentation	983

Section No.	Title	Page No.
Chapter 43 SRC: System Reset Controller Driver		
43.1	Overview	988
43.2	Macro Definition Documentation	989
43.2.1	FSL_SRC_DRIVER_VERSION	989
43.3	Enumeration Type Documentation	989
43.3.1	_src_reset_status_flags	989
43.3.2	src_warm_reset_bypass_count_t	989
43.4	Function Documentation	990
43.4.1	SRC_EnableWDOGReset	990
43.4.2	SRC_SetWarmResetBypassCount	990
43.4.3	SRC_EnableWarmReset	990
43.4.4	SRC_GetBootModeWord1	991
43.4.5	SRC_GetBootModeWord2	992
43.4.6	SRC_SetWarmBootIndication	992
43.4.7	SRC_GetResetStatusFlags	992
43.4.8	SRC_ClearResetStatusFlags	993
43.4.9	SRC_SetGeneralPurposeRegister	993
43.4.10	SRC_GetGeneralPurposeRegister	993
Chapter 44 TEMPMON: Temperature Monitor Module		
44.1	Overview	994
44.2	TEMPMON: Temperature Monitor Module	994
44.2.1	TEMPMON Operations	994
44.3	Data Structure Documentation	995
44.3.1	struct tempmon_config_t	995
44.4	Macro Definition Documentation	995
44.4.1	FSL_TEMPMON_DRIVER_VERSION	995
44.5	Enumeration Type Documentation	996
44.5.1	tempmon_alarm_mode	996
44.6	Function Documentation	996
44.6.1	TEMPMON_Init	996
44.6.2	TEMPMON_Deinit	996
44.6.3	TEMPMON_GetDefaultConfig	996
44.6.4	TEMPMON_StartMeasure	996
44.6.5	TEMPMON_StopMeasure	997
44.6.6	TEMPMON_GetCurrentTemperature	997

Section No.	Title	Page No.
44.6.7	TEMPMON_SetTempAlarm	997
Chapter 45 TRNG: True Random Number Generator		
45.1	TRNG Initialization	998
45.2	Get random data from TRNG	998
Chapter 46 USDHC: Ultra Secured Digital Host Controller Driver		
46.1	Overview	999
46.2	Typical use case	999
46.2.1	USDHC Operation	999
46.3	Data Structure Documentation	1010
46.3.1	struct usdhc_adma2_descriptor_t	1010
46.3.2	struct usdhc_capability_t	1010
46.3.3	struct usdhc_boot_config_t	1011
46.3.4	struct usdhc_config_t	1012
46.3.5	struct usdhc_command_t	1012
46.3.6	struct usdhc_adma_config_t	1013
46.3.7	struct usdhc_scatter_gather_data_list_t	1013
46.3.8	struct usdhc_scatter_gather_data_t	1014
46.3.9	struct usdhc_scatter_gather_transfer_t	1014
46.3.10	struct usdhc_data_t	1015
46.3.11	struct usdhc_transfer_t	1016
46.3.12	struct usdhc_transfer_callback_t	1016
46.3.13	struct _usdhc_handle	1017
46.3.14	struct usdhc_host_t	1018
46.4	Macro Definition Documentation	1018
46.4.1	FSL_USDHC_DRIVER_VERSION	1018
46.4.2	USDHC_ADMA1_ADDRESS_ALIGN	1018
46.4.3	USDHC_ADMA1_LENGTH_ALIGN	1018
46.4.4	USDHC_ADMA2_ADDRESS_ALIGN	1018
46.4.5	USDHC_ADMA2_LENGTH_ALIGN	1018
46.4.6	USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT	1018
46.4.7	USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK	1019
46.4.8	USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT	1019
46.4.9	USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK	1019
46.4.10	USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY	1019
46.4.11	USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT	1019
46.4.12	USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK	1019
46.4.13	USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY	1019

Section No.	Title	Page No.
46.5	Typedef Documentation	1019
46.5.1	usdhc_adma1_descriptor_t	1019
46.5.2	usdhc_transfer_function_t	1019
46.6	Enumeration Type Documentation	1020
46.6.1	anonymous enum	1020
46.6.2	anonymous enum	1020
46.6.3	anonymous enum	1021
46.6.4	anonymous enum	1021
46.6.5	anonymous enum	1021
46.6.6	anonymous enum	1022
46.6.7	anonymous enum	1022
46.6.8	anonymous enum	1023
46.6.9	anonymous enum	1024
46.6.10	anonymous enum	1024
46.6.11	anonymous enum	1024
46.6.12	anonymous enum	1024
46.6.13	usdhc_transfer_direction_t	1025
46.6.14	usdhc_data_bus_width_t	1025
46.6.15	usdhc_endian_mode_t	1025
46.6.16	usdhc_dma_mode_t	1026
46.6.17	anonymous enum	1026
46.6.18	usdhc_boot_mode_t	1026
46.6.19	usdhc_card_command_type_t	1026
46.6.20	usdhc_card_response_type_t	1026
46.6.21	anonymous enum	1027
46.6.22	anonymous enum	1027
46.6.23	anonymous enum	1028
46.6.24	usdhc_burst_len_t	1028
46.6.25	anonymous enum	1028
46.7	Function Documentation	1028
46.7.1	USDHC_Init	1028
46.7.2	USDHC_Deinit	1029
46.7.3	USDHC_Reset	1029
46.7.4	USDHC_SetAdmaTableConfig	1029
46.7.5	USDHC_SetInternalDmaConfig	1030
46.7.6	USDHC_SetADMA2Descriptor	1030
46.7.7	USDHC_SetADMA1Descriptor	1031
46.7.8	USDHC_EnableInternalDMA	1031
46.7.9	USDHC_EnableInterruptStatus	1032
46.7.10	USDHC_DisableInterruptStatus	1032
46.7.11	USDHC_EnableInterruptSignal	1032
46.7.12	USDHC_DisableInterruptSignal	1032
46.7.13	USDHC_GetEnabledInterruptStatusFlags	1033

Section No.	Title	Page No.
46.7.14	USDHC_GetInterruptStatusFlags	1033
46.7.15	USDHC_ClearInterruptStatusFlags	1033
46.7.16	USDHC_GetAutoCommand12ErrorStatusFlags	1034
46.7.17	USDHC_GetAdmaErrorStatusFlags	1035
46.7.18	USDHC_GetPresentStatusFlags	1035
46.7.19	USDHC_GetCapability	1035
46.7.20	USDHC_ForceClockOn	1036
46.7.21	USDHC_SetSdClock	1036
46.7.22	USDHC_SetCardActive	1036
46.7.23	USDHC_AssertHardwareReset	1037
46.7.24	USDHC_SetDataBusWidth	1037
46.7.25	USDHC_WriteData	1037
46.7.26	USDHC_ReadData	1037
46.7.27	USDHC_SendCommand	1038
46.7.28	USDHC_EnableWakeupEvent	1038
46.7.29	USDHC_CardDetectByData3	1038
46.7.30	USDHC_DetectCardInsert	1038
46.7.31	USDHC_EnableSdioControl	1039
46.7.32	USDHC_SetContinueRequest	1039
46.7.33	USDHC_RequestStopAtBlockGap	1039
46.7.34	USDHC_SetMmcBootConfig	1039
46.7.35	USDHC_EnableMmcBoot	1040
46.7.36	USDHC_SetForceEvent	1040
46.7.37	USDHC_SelectVoltage	1040
46.7.38	USDHC_RequestTuningForSDR50	1040
46.7.39	USDHC_RequestReTuning	1041
46.7.40	USDHC_EnableAutoTuning	1041
46.7.41	USDHC_EnableAutoTuningForCmdAndData	1041
46.7.42	USDHC_EnableManualTuning	1041
46.7.43	USDHC_GetTuningDelayStatus	1042
46.7.44	USDHC_SetTuningDelay	1042
46.7.45	USDHC_AdjustDelayForManualTuning	1042
46.7.46	USDHC_SetStandardTuningCounter	1043
46.7.47	USDHC_EnableStandardTuning	1043
46.7.48	USDHC_GetExecuteStdTuningStatus	1044
46.7.49	USDHC_CheckStdTuningResult	1045
46.7.50	USDHC_CheckTuningError	1045
46.7.51	USDHC_EnableDDRMode	1045
46.7.52	USDHC_EnableHS400Mode	1045
46.7.53	USDHC_ResetStrobeDLL	1046
46.7.54	USDHC_EnableStrobeDLL	1046
46.7.55	USDHC_ConfigStrobeDLL	1046
46.7.56	USDHC_SetStrobeDllOverride	1046
46.7.57	USDHC_GetStrobeDLLStatus	1047
46.7.58	USDHC_SetDataConfig	1047

Section No.	Title	Page No.
46.7.59	USDHC_TransferCreateHandle	1047
46.7.60	USDHC_TransferNonBlocking	1048
46.7.61	USDHC_TransferBlocking	1048
46.7.62	USDHC_TransferHandleIRQ	1049

Chapter 47 WDOG: Watchdog Timer Driver

47.1	Overview	1050
47.2	Typical use case	1050
47.3	Data Structure Documentation	1051
47.3.1	struct wdog_work_mode_t	1051
47.3.2	struct wdog_config_t	1051
47.4	Enumeration Type Documentation	1052
47.4.1	_wdog_interrupt_enable	1052
47.4.2	_wdog_status_flags	1052
47.5	Function Documentation	1052
47.5.1	WDOG_GetDefaultConfig	1052
47.5.2	WDOG_Init	1053
47.5.3	WDOG_Deinit	1053
47.5.4	WDOG_Enable	1053
47.5.5	WDOG_Disable	1054
47.5.6	WDOG_TriggerSystemSoftwareReset	1054
47.5.7	WDOG_TriggerSoftwareSignal	1054
47.5.8	WDOG_EnableInterrupts	1055
47.5.9	WDOG_GetStatusFlags	1056
47.5.10	WDOG_ClearInterruptStatus	1056
47.5.11	WDOG_SetTimeoutValue	1057
47.5.12	WDOG_SetInterruptTimeoutValue	1057
47.5.13	WDOG_DisablePowerDownEnable	1057
47.5.14	WDOG_Refresh	1058

Chapter 48 XBARA: Inter-Peripheral Crossbar Switch

48.1	Overview	1059
48.2	Function	1059
48.2.1	XBARA Initialization	1059
48.2.2	Call diagram	1059
48.3	Typical use case	1059
48.4	Data Structure Documentation	1060

Section No.	Title	Page No.
48.4.1	struct xbara_control_config_t	1060
48.5	Enumeration Type Documentation	1061
48.5.1	xbara_active_edge_t	1061
48.5.2	xbara_request_t	1061
48.5.3	xbara_status_flag_t	1061
48.6	Function Documentation	1062
48.6.1	XBARA_Init	1062
48.6.2	XBARA_Deinit	1063
48.6.3	XBARA_SetSignalsConnection	1063
48.6.4	XBARA_GetStatusFlags	1063
48.6.5	XBARA_ClearStatusFlags	1064
48.6.6	XBARA_SetOutputSignalConfig	1064
 Chapter 49 XBARB: Inter-Peripheral Crossbar Switch		
49.1	Overview	1065
49.2	Function groups	1065
49.2.1	XBARB Initialization	1065
49.2.2	Call diagram	1065
49.3	Typical use case	1065
49.4	Function Documentation	1066
49.4.1	XBARB_Init	1066
49.4.2	XBARB_Deinit	1067
49.4.3	XBARB_SetSignalsConnection	1067
 Chapter 50 Debug Console		
50.1	Overview	1068
50.2	Function groups	1068
50.2.1	Initialization	1068
50.2.2	Advanced Feature	1069
50.2.3	SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART	1073
50.3	Typical use case	1074
50.4	Macro Definition Documentation	1076
50.4.1	DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN	1076
50.4.2	DEBUGCONSOLE_REDIRECT_TO_SDK	1076
50.4.3	DEBUGCONSOLE_DISABLE	1076
50.4.4	SDK_DEBUGCONSOLE	1076

Section No.	Title	Page No.
50.4.5	PRINTF	1076
50.5	Function Documentation	1076
50.5.1	DbgConsole_Init	1076
50.5.2	DbgConsole_Deinit	1077
50.5.3	DbgConsole_EnterLowpower	1077
50.5.4	DbgConsole_ExitLowpower	1078
50.5.5	DbgConsole_Printf	1078
50.5.6	DbgConsole_Vprintf	1078
50.5.7	DbgConsole_Putchar	1078
50.5.8	DbgConsole_Scanf	1079
50.5.9	DbgConsole_Getchar	1079
50.5.10	DbgConsole_BlockingPrintf	1080
50.5.11	DbgConsole_BlockingVprintf	1080
50.5.12	DbgConsole_Flush	1080
50.5.13	DbgConsole_TryGetchar	1081
50.6	debug console configuration	1083
50.6.1	Overview	1083
50.6.2	Macro Definition Documentation	1084
50.7	Semihosting	1086
50.7.1	Guide Semihosting for IAR	1086
50.7.2	Guide Semihosting for Keil μ Vision	1086
50.7.3	Guide Semihosting for MCUXpresso IDE	1087
50.7.4	Guide Semihosting for ARMGCC	1087
50.8	SWO	1090
50.8.1	Guide SWO for SDK	1090
50.8.2	Guide SWO for Keil μ Vision	1091
50.8.3	Guide SWO for MCUXpresso IDE	1092
50.8.4	Guide SWO for ARMGCC	1092
 Chapter 51 Notification Framework		
51.1	Overview	1093
51.2	Notifier Overview	1093
51.3	Data Structure Documentation	1095
51.3.1	struct notifier_notification_block_t	1095
51.3.2	struct notifier_callback_config_t	1096
51.3.3	struct notifier_handle_t	1096
51.4	Typedef Documentation	1097
51.4.1	notifier_user_config_t	1097

Section No.	Title	Page No.
51.4.2	notifier_user_function_t	1097
51.4.3	notifier_callback_t	1098
51.5	Enumeration Type Documentation	1098
51.5.1	_notifier_status	1098
51.5.2	notifier_policy_t	1099
51.5.3	notifier_notification_type_t	1099
51.5.4	notifier_callback_type_t	1099
51.6	Function Documentation	1100
51.6.1	NOTIFIER_CreateHandle	1100
51.6.2	NOTIFIER_SwitchConfig	1101
51.6.3	NOTIFIER_GetErrorCallbackIndex	1102
 Chapter 52 Shell		
52.1	Overview	1103
52.2	Function groups	1103
52.2.1	Initialization	1103
52.2.2	Advanced Feature	1103
52.2.3	Shell Operation	1103
52.3	Data Structure Documentation	1105
52.3.1	struct shell_command_t	1105
52.4	Macro Definition Documentation	1106
52.4.1	SHELL_NON_BLOCKING_MODE	1106
52.4.2	SHELL_AUTO_COMPLETE	1106
52.4.3	SHELL_BUFFER_SIZE	1106
52.4.4	SHELL_MAX_ARGS	1106
52.4.5	SHELL_HISTORY_COUNT	1106
52.4.6	SHELL_HANDLE_SIZE	1106
52.4.7	SHELL_USE_COMMON_TASK	1107
52.4.8	SHELL_TASK_PRIORITY	1107
52.4.9	SHELL_TASK_STACK_SIZE	1107
52.4.10	SHELL_HANDLE_DEFINE	1107
52.4.11	SHELL_COMMAND_DEFINE	1107
52.4.12	SHELL_COMMAND	1108
52.5	Typedef Documentation	1108
52.5.1	cmd_function_t	1108
52.6	Enumeration Type Documentation	1108
52.6.1	shell_status_t	1108

Section No.	Title	Page No.
52.7	Function Documentation	1108
52.7.1	SHELL_Init	1108
52.7.2	SHELL_RegisterCommand	1109
52.7.3	SHELL_UnregisterCommand	1110
52.7.4	SHELL_Write	1110
52.7.5	SHELL_Printf	1111
52.7.6	SHELL_WriteSynchronization	1111
52.7.7	SHELL_PrintfSynchronization	1111
52.7.8	SHELL_ChangePrompt	1112
52.7.9	SHELL_PrintPrompt	1112
52.7.10	SHELL_Task	1112
52.7.11	SHELL_checkRunningInIsr	1113
Chapter 53	Cards: Secure Digital Card/Embedded MultiMedia Card/SDIO Card	
53.1	Overview	1114
53.2	SDIO Card Driver	1115
53.2.1	Overview	1115
53.2.2	SDIO CARD Operation	1115
53.2.3	Data Structure Documentation	1117
53.2.4	Macro Definition Documentation	1119
53.2.5	Enumeration Type Documentation	1119
53.2.6	Function Documentation	1119
53.3	SD Card Driver	1136
53.3.1	Overview	1136
53.3.2	SD CARD Operation	1136
53.3.3	Data Structure Documentation	1139
53.3.4	Macro Definition Documentation	1140
53.3.5	Enumeration Type Documentation	1140
53.3.6	Function Documentation	1140
53.4	MMC Card Driver	1150
53.4.1	Overview	1150
53.4.2	MMC CARD Operation	1150
53.4.3	Data Structure Documentation	1152
53.4.4	Macro Definition Documentation	1154
53.4.5	Enumeration Type Documentation	1154
53.4.6	Function Documentation	1155
53.5	SDMMC HOST Driver	1168
53.5.1	Overview	1168
53.6	SDMMC OSA	1169

Section No.	Title	Page No.
53.6.1	Overview	1169
53.6.2	Data Structure Documentation	1170
53.6.3	Function Documentation	1170
53.6.4	USDHC HOST adapter Driver	1174
53.7	SDMMC Common	1187
53.7.1	Overview	1187
53.7.2	Data Structure Documentation	1206
53.7.3	Macro Definition Documentation	1219
53.7.4	Enumeration Type Documentation	1219
53.7.5	Function Documentation	1237
 Chapter 54 CODEC Driver		
54.1	Overview	1241
54.2	CODEC Common Driver	1242
54.2.1	Overview	1242
54.2.2	Data Structure Documentation	1247
54.2.3	Macro Definition Documentation	1248
54.2.4	Enumeration Type Documentation	1248
54.2.5	Function Documentation	1253
54.3	CODEC I2C Driver	1257
54.3.1	Overview	1257
54.3.2	Data Structure Documentation	1258
54.3.3	Enumeration Type Documentation	1258
54.3.4	Function Documentation	1258
54.4	CS42888 Driver	1261
54.4.1	Overview	1261
54.4.2	Data Structure Documentation	1263
54.4.3	Macro Definition Documentation	1264
54.4.4	Enumeration Type Documentation	1264
54.4.5	Function Documentation	1265
54.4.6	CS42888 Adapter	1271
54.5	DA7212 Driver	1279
54.5.1	Overview	1279
54.5.2	Data Structure Documentation	1282
54.5.3	Macro Definition Documentation	1283
54.5.4	Enumeration Type Documentation	1283
54.5.5	Function Documentation	1285
54.5.6	DA7212 Adapter	1290
54.6	SGTL5000 Driver	1298

Section No.	Title	Page No.
54.6.1	Overview	1298
54.6.2	Data Structure Documentation	1300
54.6.3	Macro Definition Documentation	1301
54.6.4	Enumeration Type Documentation	1301
54.6.5	Function Documentation	1303
54.6.6	SGTL5000 Adapter	1309
54.7	WM8960 Driver	1317
54.7.1	Overview	1317
54.7.2	Data Structure Documentation	1320
54.7.3	Macro Definition Documentation	1322
54.7.4	Enumeration Type Documentation	1322
54.7.5	Function Documentation	1324
54.7.6	WM8960 Adapter	1331
54.8	WM8904 Driver	1339
54.8.1	Overview	1339
54.8.2	Data Structure Documentation	1343
54.8.3	Macro Definition Documentation	1344
54.8.4	Enumeration Type Documentation	1344
54.8.5	Function Documentation	1347
54.8.6	WM8904 Adapter	1356
 Chapter 55 Serial Manager		
55.1	Overview	1364
55.2	Data Structure Documentation	1367
55.2.1	struct serial_manager_config_t	1367
55.2.2	struct serial_manager_callback_message_t	1368
55.3	Macro Definition Documentation	1368
55.3.1	SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE	1368
55.3.2	SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE	1368
55.3.3	SERIAL_MANAGER_USE_COMMON_TASK	1368
55.3.4	SERIAL_MANAGER_HANDLE_SIZE	1368
55.3.5	SERIAL_MANAGER_HANDLE_DEFINE	1368
55.3.6	SERIAL_MANAGER_WRITE_HANDLE_DEFINE	1369
55.3.7	SERIAL_MANAGER_READ_HANDLE_DEFINE	1369
55.3.8	SERIAL_MANAGER_TASK_PRIORITY	1370
55.3.9	SERIAL_MANAGER_TASK_STACK_SIZE	1370
55.4	Enumeration Type Documentation	1370
55.4.1	serial_port_type_t	1370
55.4.2	serial_manager_type_t	1370

Section No.	Title	Page No.
55.4.3	serial_manager_status_t	1370
55.5	Function Documentation	1371
55.5.1	SerialManager_Init	1371
55.5.2	SerialManager_Deinit	1372
55.5.3	SerialManager_OpenWriteHandle	1372
55.5.4	SerialManager_CloseWriteHandle	1374
55.5.5	SerialManager_OpenReadHandle	1375
55.5.6	SerialManager_CloseReadHandle	1376
55.5.7	SerialManager_WriteBlocking	1376
55.5.8	SerialManager_ReadBlocking	1377
55.5.9	SerialManager_WriteNonBlocking	1378
55.5.10	SerialManager_ReadNonBlocking	1378
55.5.11	SerialManager_TryRead	1379
55.5.12	SerialManager_CancelWriting	1380
55.5.13	SerialManager_CancelReading	1380
55.5.14	SerialManager_InstallTxCallback	1381
55.5.15	SerialManager_InstallRxCallback	1381
55.5.16	SerialManager_needPollingIsr	1383
55.5.17	SerialManager_EnterLowpower	1383
55.5.18	SerialManager_ExitLowpower	1383
55.5.19	SerialManager_SetLowpowerCriticalCb	1384
55.6	Serial Port Uart	1385
55.6.1	Overview	1385
55.6.2	Enumeration Type Documentation	1385
55.7	Serial Port USB	1386
55.7.1	Overview	1386
55.7.2	Data Structure Documentation	1387
55.7.3	Enumeration Type Documentation	1387
55.7.4	USB Device Configuration	1388
55.8	Serial Port SWO	1389
55.8.1	Overview	1389
55.8.2	Data Structure Documentation	1389
55.8.3	Enumeration Type Documentation	1389
 Chapter 56 Nic301		
56.1	Overview	1390
56.2	Macro Definition Documentation	1390
56.2.1	FSL_NIC301_DRIVER_VERSION	1390
56.3	Function Documentation	1390



Section No.	Title	Page No.
56.3.1	NIC_SetReadQos	1390
56.3.2	NIC_GetReadQos	1391
56.3.3	NIC_SetWriteQos	1391
56.3.4	NIC_GetWriteQos	1391
56.3.5	NIC_SetFnModAhb	1391
56.3.6	NIC_GetFnModAhb	1392
56.3.7	NIC_SetWrTideMark	1392
56.3.8	NIC_GetWrTideMark	1392
56.3.9	NIC_SetFnMod	1392
56.3.10	NIC_GetFnMod	1393
56.3.11	NIC_SetFnMod2	1393
56.3.12	NIC_GetFnMod2	1393
56.3.13	CODEC Adapter	1394

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.

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

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

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

MCUXpresso SDK Folder Structure

Chapter 2

Trademarks

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

How to Reach Us:

Home Page: nxp.com

Web Support: nxp.com/support

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

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, 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, Arm7TD-MI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, Vision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2021 NXP B.V.

Chapter 3

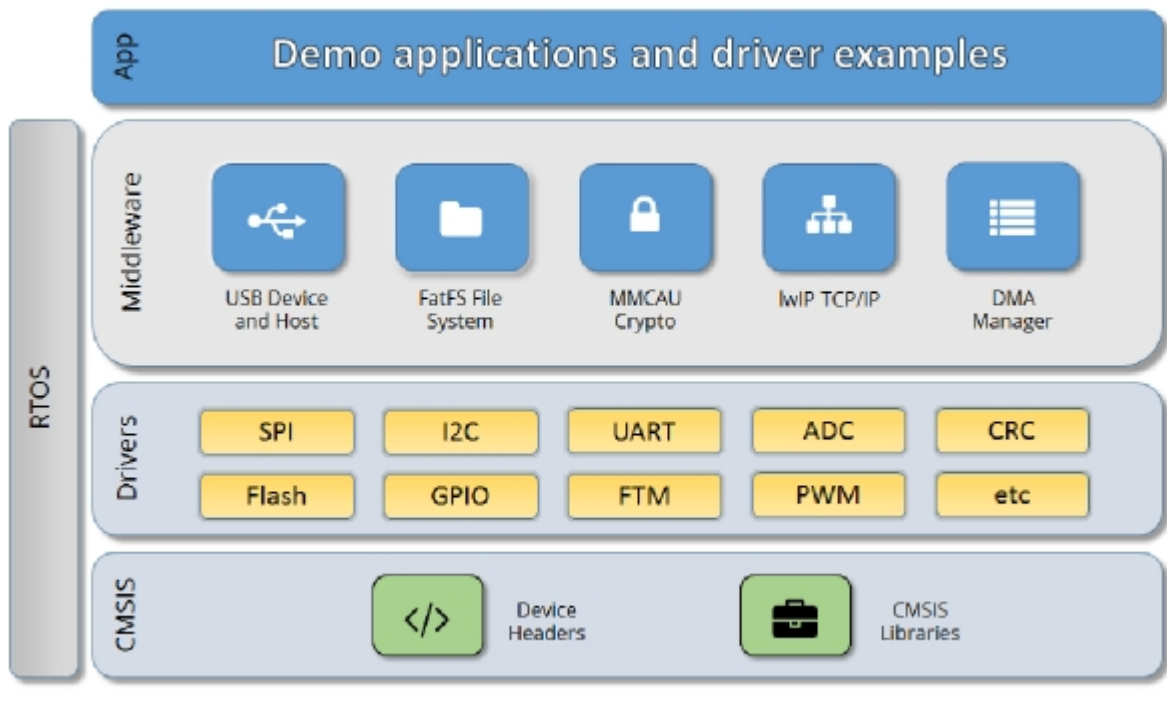
Architectural Overview

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

Overview

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

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



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

Clock Driver

4.1 Overview

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

The clock driver supports:

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

Files

- file [fsl_clock.h](#)

Data Structures

- struct [clock_usb_pll_config_t](#)
PLL configuration for USB. [More...](#)
- struct [clock_sys_pll_config_t](#)
PLL configuration for System. [More...](#)
- struct [clock_audio_pll_config_t](#)
PLL configuration for AUDIO and VIDEO. [More...](#)
- struct [clock_enet_pll_config_t](#)
PLL configuration for ENET. [More...](#)

Macros

- #define [FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL](#) 0
Configure whether driver controls clock.
- #define [CCSR_OFFSET](#) 0x0C
CCM registers offset.
- #define [PLL_SYS_OFFSET](#) 0x30
CCM Analog registers offset.
- #define [CCM_ANALOG_TUPLE](#)(reg, shift) (((reg)&0xFFFU) << 16U) | (shift)
CCM ANALOG tuple macros to map corresponding registers and bit fields.
- #define [CLKPN_FREQ](#) 0U
clockIPN frequency.
- #define [ADC_CLOCKS](#)
Clock ip name array for ADC.
- #define [AOI_CLOCKS](#)
Clock ip name array for AOI.
- #define [BEE_CLOCKS](#)
Clock ip name array for BEE.
- #define [CMP_CLOCKS](#)

- *Clock ip name array for CMP.*
• #define **DCDC_CLOCKS**
- *Clock ip name array for DCDC.*
• #define **DCP_CLOCKS**
- *Clock ip name array for DCP.*
• #define **DMAMUX_CLOCKS**
- *Clock ip name array for DMAMUX_CLOCKS.*
• #define **EDMA_CLOCKS**
- *Clock ip name array for DMA.*
• #define **ENC_CLOCKS**
- *Clock ip name array for ENC.*
• #define **ENET_CLOCKS**
- *Clock ip name array for ENET.*
• #define **EWM_CLOCKS**
- *Clock ip name array for EWM.*
• #define **FLEXCAN_CLOCKS**
- *Clock ip name array for FLEXCAN.*
• #define **FLEXCAN_PERIPH_CLOCKS**
- *Clock ip name array for FLEXCAN Peripheral clock.*
• #define **FLEXIO_CLOCKS**
- *Clock ip name array for FLEXIO.*
• #define **FLEXRAM_CLOCKS**
- *Clock ip name array for FLEXRAM.*
• #define **FLEXSPI_CLOCKS**
- *Clock ip name array for FLEXSPI.*
• #define **FLEXSPI_EXSC_CLOCKS**
- *Clock ip name array for FLEXSPI EXSC.*
• #define **GPIO_CLOCKS**
- *Clock ip name array for GPIO.*
• #define **GPT_CLOCKS**
- *Clock ip name array for GPT.*
• #define **KPP_CLOCKS**
- *Clock ip name array for KPP.*
• #define **LPI2C_CLOCKS**
- *Clock ip name array for LPI2C.*
• #define **LPSPI_CLOCKS**
- *Clock ip name array for LPSPI.*
• #define **LPUART_CLOCKS**
- *Clock ip name array for LPUART.*
• #define **OCRAM_EXSC_CLOCKS**
- *Clock ip name array for OCRAM EXSC.*
• #define **PIT_CLOCKS**
- *Clock ip name array for PIT.*
• #define **PWM_CLOCKS**
- *Clock ip name array for PWM.*
• #define **RTWDOG_CLOCKS**
- *Clock ip name array for RTWDOG.*
• #define **SAI_CLOCKS**
- *Clock ip name array for SAI.*
• #define **SEMC_CLOCKS**
- *Clock ip name array for SEMC.*

- #define `SEMC_EXSC_CLOCKS`
Clock ip name array for SEMC EXSC.
- #define `TMR_CLOCKS`
Clock ip name array for QTIMER.
- #define `TRNG_CLOCKS`
Clock ip name array for TRNG.
- #define `WDOG_CLOCKS`
Clock ip name array for WDOG.
- #define `USDHC_CLOCKS`
Clock ip name array for USDHC.
- #define `SPDIF_CLOCKS`
Clock ip name array for SPDIF.
- #define `XBARA_CLOCKS`
Clock ip name array for XBARA.
- #define `XBARB_CLOCKS`
Clock ip name array for XBARB.
- #define `kCLOCK_CoreSysClk kCLOCK_CpuClk`
For compatible with other platforms without CCM.
- #define `CLOCK_GetCoreSysClkFreq CLOCK_GetCpuClkFreq`
For compatible with other platforms without CCM.

Enumerations

- enum `clock_name_t` {
`kCLOCK_CpuClk` = 0x0U,
`kCLOCK_AhbClk` = 0x1U,
`kCLOCK_SemcClk` = 0x2U,
`kCLOCK_IpgClk` = 0x3U,
`kCLOCK_PerClk` = 0x4U,
`kCLOCK_OscClk` = 0x5U,
`kCLOCK_RtcClk` = 0x6U,
`kCLOCK_Usb1PllClk` = 0x7U,
`kCLOCK_Usb1PllPfd0Clk` = 0x8U,
`kCLOCK_Usb1PllPfd1Clk` = 0x9U,
`kCLOCK_Usb1PllPfd2Clk` = 0xAU,
`kCLOCK_Usb1PllPfd3Clk` = 0xBU,
`kCLOCK_Usb1SwClk` = 0x15U,
`kCLOCK_Usb1Sw60MClk` = 0x16U,
`kCLOCK_Usb1Sw80MClk` = 0x1BU,
`kCLOCK_SysPllClk` = 0xCU,
`kCLOCK_SysPllPfd0Clk` = 0xDU,
`kCLOCK_SysPllPfd1Clk` = 0xEU,
`kCLOCK_SysPllPfd2Clk` = 0xFU,
`kCLOCK_SysPllPfd3Clk` = 0x10U,
`kCLOCK_EnetPllClk` = 0x11U,
`kCLOCK_EnetPll25MClk` = 0x12U,
`kCLOCK_EnetPll500MClk` = 0x13U,
`kCLOCK_AudioPllClk` = 0x14U,

```
kCLOCK_NoneName = CLOCK_SOURCE_NONE }
```

Clock name used to get clock frequency.

- enum `clock_ip_name_t` { ,

kCLOCK_Aips_tz1 = (0U << 8U) | CCM_CCGR0_CG0_SHIFT,
 kCLOCK_Aips_tz2 = (0U << 8U) | CCM_CCGR0_CG1_SHIFT,
 kCLOCK_Mqs = (0U << 8U) | CCM_CCGR0_CG2_SHIFT,
 kCLOCK_Sim_m_clk_r = (0U << 8U) | CCM_CCGR0_CG4_SHIFT,
 kCLOCK_Dcp = (0U << 8U) | CCM_CCGR0_CG5_SHIFT,
 kCLOCK_Lpuart3 = (0U << 8U) | CCM_CCGR0_CG6_SHIFT,
 kCLOCK_Can1 = (0U << 8U) | CCM_CCGR0_CG7_SHIFT,
 kCLOCK_Can1S = (0U << 8U) | CCM_CCGR0_CG8_SHIFT,
 kCLOCK_Can2 = (0U << 8U) | CCM_CCGR0_CG9_SHIFT,
 kCLOCK_Can2S = (0U << 8U) | CCM_CCGR0_CG10_SHIFT,
 kCLOCK_Trace = (0U << 8U) | CCM_CCGR0_CG11_SHIFT,
 kCLOCK_Gpt2 = (0U << 8U) | CCM_CCGR0_CG12_SHIFT,
 kCLOCK_Gpt2S = (0U << 8U) | CCM_CCGR0_CG13_SHIFT,
 kCLOCK_Lpuart2 = (0U << 8U) | CCM_CCGR0_CG14_SHIFT,
 kCLOCK_Gpio2 = (0U << 8U) | CCM_CCGR0_CG15_SHIFT,
 kCLOCK_Lpspi1 = (1U << 8U) | CCM_CCGR1_CG0_SHIFT,
 kCLOCK_Lpspi2 = (1U << 8U) | CCM_CCGR1_CG1_SHIFT,
 kCLOCK_Lpspi3 = (1U << 8U) | CCM_CCGR1_CG2_SHIFT,
 kCLOCK_Lpspi4 = (1U << 8U) | CCM_CCGR1_CG3_SHIFT,
 kCLOCK_Adc2 = (1U << 8U) | CCM_CCGR1_CG4_SHIFT,
 kCLOCK_Enet = (1U << 8U) | CCM_CCGR1_CG5_SHIFT,
 kCLOCK_Pit = (1U << 8U) | CCM_CCGR1_CG6_SHIFT,
 kCLOCK_Adc1 = (1U << 8U) | CCM_CCGR1_CG8_SHIFT,
 kCLOCK_SemcExsc = (1U << 8U) | CCM_CCGR1_CG9_SHIFT,
 kCLOCK_Gpt1 = (1U << 8U) | CCM_CCGR1_CG10_SHIFT,
 kCLOCK_Gpt1S = (1U << 8U) | CCM_CCGR1_CG11_SHIFT,
 kCLOCK_Lpuart4 = (1U << 8U) | CCM_CCGR1_CG12_SHIFT,
 kCLOCK_Gpio1 = (1U << 8U) | CCM_CCGR1_CG13_SHIFT,
 kCLOCK_Csu = (1U << 8U) | CCM_CCGR1_CG14_SHIFT,
 kCLOCK_Gpio5 = (1U << 8U) | CCM_CCGR1_CG15_SHIFT,
 kCLOCK_OcramExsc = (2U << 8U) | CCM_CCGR2_CG0_SHIFT,
 kCLOCK_IomuxcSnvs = (2U << 8U) | CCM_CCGR2_CG2_SHIFT,
 kCLOCK_Lpi2c1 = (2U << 8U) | CCM_CCGR2_CG3_SHIFT,
 kCLOCK_Lpi2c2 = (2U << 8U) | CCM_CCGR2_CG4_SHIFT,
 kCLOCK_Lpi2c3 = (2U << 8U) | CCM_CCGR2_CG5_SHIFT,
 kCLOCK_Ocotp = (2U << 8U) | CCM_CCGR2_CG6_SHIFT,
 kCLOCK_Xbar1 = (2U << 8U) | CCM_CCGR2_CG11_SHIFT,
 kCLOCK_Xbar2 = (2U << 8U) | CCM_CCGR2_CG12_SHIFT,
 kCLOCK_Gpio3 = (2U << 8U) | CCM_CCGR2_CG13_SHIFT,
 kCLOCK_Lpuart5 = (3U << 8U) | CCM_CCGR3_CG1_SHIFT,
 kCLOCK_Semc = (3U << 8U) | CCM_CCGR3_CG2_SHIFT,
 kCLOCK_Lpuart6 = (3U << 8U) | CCM_CCGR3_CG3_SHIFT,
 kCLOCK_Aoi = (3U << 8U) | CCM_CCGR3_CG4_SHIFT,
 kCLOCK_Ewm0 = (3U << 8U) | CCM_CCGR3_CG7_SHIFT,
 kCLOCK_Wdog1 = (3U << 8U) | CCM_CCGR3_CG8_SHIFT,
 kCLOCK_FlexRam = (3U << 8U) | CCM_CCGR3_CG9_SHIFT,
 kCLOCK_Acomp1 = (3U << 8U) | CCM_CCGR3_CG10_SHIFT,
 kCLOCK_Acomp2 = (3U << 8U) | CCM_CCGR3_CG11_SHIFT,
 kCLOCK_Acomp3 = (3U << 8U) | CCM_CCGR3_CG12_SHIFT,

```
kCLOCK_Timer2 = (6U << 8U) | CCM_CCGR6_CG14_SHIFT }
```

CCM CCGR gate control for each module independently.

- enum `clock_osc_t` {
`kCLOCK_RcOsc = 0U,`
`kCLOCK_XtalOsc = 1U }`

OSC 24M source select.
- enum `clock_gate_value_t` {
`kCLOCK_ClockNotNeeded = 0U,`
`kCLOCK_ClockNeededRun = 1U,`
`kCLOCK_ClockNeededRunWait = 3U }`

Clock gate value.
- enum `clock_mode_t` {
`kCLOCK_ModeRun = 0U,`
`kCLOCK_ModeWait = 1U,`
`kCLOCK_ModeStop = 2U }`

System clock mode.
- enum `clock_mux_t` {
`kCLOCK_Pll3SwMux,`
`kCLOCK_PeriphMux,`
`kCLOCK_SemcAltMux,`
`kCLOCK_SemcMux,`
`kCLOCK_PrePeriphMux,`
`kCLOCK_TraceMux,`
`kCLOCK_PeriphClk2Mux,`
`kCLOCK_LpspiMux,`
`kCLOCK_FlexspiMux,`
`kCLOCK_Usdhc2Mux,`
`kCLOCK_Usdhc1Mux,`
`kCLOCK_Sai3Mux,`
`kCLOCK_Sai2Mux,`
`kCLOCK_Sai1Mux,`
`kCLOCK_PerclkMux,`
`kCLOCK_Flexio1Mux,`
`kCLOCK_CanMux,`
`kCLOCK_UartMux,`
`kCLOCK_SpdifMux,`
`kCLOCK_Lpi2cMux }`

MUX control names for clock mux setting.
- enum `clock_div_t` {

```
kCLOCK_ArmDiv,  
kCLOCK_PeriphClk2Div,  
kCLOCK_SemcDiv,  
kCLOCK_AhbDiv,  
kCLOCK_IpgDiv,  
kCLOCK_LpspiDiv,  
kCLOCK_FlexspiDiv,  
kCLOCK_PerclkDiv,  
kCLOCK_CanDiv,  
kCLOCK_TraceDiv,  
kCLOCK_Usdhc2Div,  
kCLOCK_Usdhc1Div,  
kCLOCK_UartDiv,  
kCLOCK_Flexio1Div,  
kCLOCK_Sai3PreDiv,  
kCLOCK_Sai3Div,  
kCLOCK_Flexio1PreDiv,  
kCLOCK_Sai1PreDiv,  
kCLOCK_Sai1Div,  
kCLOCK_Sai2PreDiv,  
kCLOCK_Sai2Div,  
kCLOCK_Spdif0PreDiv,  
kCLOCK_Spdif0Div,  
kCLOCK_Lpi2cDiv,  
kCLOCK_NonePreDiv = CLOCK_ROOT_NONE_PRE_DIV }
```

DIV control names for clock div setting.

- enum `clock_div_value_t` {

kCLOCK_ArmDivBy1 = 0,
 kCLOCK_ArmDivBy2 = 1,
 kCLOCK_ArmDivBy3 = 2,
 kCLOCK_ArmDivBy4 = 3,
 kCLOCK_ArmDivBy5 = 4,
 kCLOCK_ArmDivBy6 = 5,
 kCLOCK_ArmDivBy7 = 6,
 kCLOCK_ArmDivBy8 = 7,
 kCLOCK_PeriphClk2DivBy1 = 0,
 kCLOCK_PeriphClk2DivBy2 = 1,
 kCLOCK_PeriphClk2DivBy3 = 2,
 kCLOCK_PeriphClk2DivBy4 = 3,
 kCLOCK_PeriphClk2DivBy5 = 4,
 kCLOCK_PeriphClk2DivBy6 = 5,
 kCLOCK_PeriphClk2DivBy7 = 6,
 kCLOCK_PeriphClk2DivBy8 = 7,
 kCLOCK_SemcDivBy1 = 0,
 kCLOCK_SemcDivBy2 = 1,
 kCLOCK_SemcDivBy3 = 2,
 kCLOCK_SemcDivBy4 = 3,
 kCLOCK_SemcDivBy5 = 4,
 kCLOCK_SemcDivBy6 = 5,
 kCLOCK_SemcDivBy7 = 6,
 kCLOCK_SemcDivBy8 = 7,
 kCLOCK_AhbDivBy1 = 0,
 kCLOCK_AhbDivBy2 = 1,
 kCLOCK_AhbDivBy3 = 2,
 kCLOCK_AhbDivBy4 = 3,
 kCLOCK_AhbDivBy5 = 4,
 kCLOCK_AhbDivBy6 = 5,
 kCLOCK_AhbDivBy7 = 6,
 kCLOCK_AhbDivBy8 = 7,
 kCLOCK_IpgDivBy1 = 0,
 kCLOCK_IpgDivBy2 = 1,
 kCLOCK_IpgDivBy3 = 2,
 kCLOCK_IpgDivBy4 = 3,
 kCLOCK_LpspiDivBy1 = 0,
 kCLOCK_LpspiDivBy2 = 1,
 kCLOCK_LpspiDivBy3 = 2,
 kCLOCK_LpspiDivBy4 = 3,
 kCLOCK_LpspiDivBy5 = 4,
 kCLOCK_LpspiDivBy6 = 5,
 kCLOCK_LpspiDivBy7 = 6,
 kCLOCK_LpspiDivBy8 = 7,
 kCLOCK_FlexspiDivBy1 = 0,
 kCLOCK_FlexspiDivBy2 = 1,
 kCLOCK_FlexspiDivBy3 = 2,
 kCLOCK_FlexspiDivBy4 = 3,
 kCLOCK_FlexspiDivBy5 = 4,

- ```
kCLOCK_MiscDivBy64 = 63 }
```
- Clock divider value.*
- enum `clock_usb_src_t` {  
`kCLOCK_Usb480M` = 0,  
`kCLOCK_UsbSrcUnused` = (int)0xFFFFFFFFFU }
- USB clock source definition.*
- enum `clock_usb_phy_src_t` { `kCLOCK_Usbphy480M` = 0 }
- Source of the USB HS PHY.*
- enum `_clock_pll_clk_src` {  
`kCLOCK_PllClkSrc24M` = 0U,  
`kCLOCK_PllSrcClkPN` = 1U }
- PLL clock source, bypass cloco source also.*
- enum `clock_pll_t` {  
`kCLOCK_PllSys` = CCM\_ANALOG\_TUPLE(PLL\_SYS\_OFFSET, CCM\_ANALOG\_PLL\_SYS\_ENABLE\_SHIFT),  
`kCLOCK_PllUsb1` = CCM\_ANALOG\_TUPLE(PLL\_USB1\_OFFSET, CCM\_ANALOG\_PLL\_USB1\_ENABLE\_SHIFT),  
`kCLOCK_PllAudio` = CCM\_ANALOG\_TUPLE(PLL\_AUDIO\_OFFSET, CCM\_ANALOG\_PLL\_AUDIO\_ENABLE\_SHIFT),  
`kCLOCK_PllEnet` = CCM\_ANALOG\_TUPLE(PLL\_ENET\_OFFSET, CCM\_ANALOG\_PLL\_ENET\_ENABLE\_SHIFT),  
`kCLOCK_PllEnet500M` = CCM\_ANALOG\_TUPLE(PLL\_ENET\_OFFSET, CCM\_ANALOG\_PLL\_ENET\_ENET\_500M\_REF\_EN\_SHIFT),  
`kCLOCK_PllEnet25M` = CCM\_ANALOG\_TUPLE(PLL\_ENET\_OFFSET, CCM\_ANALOG\_PLL\_ENET\_ENET\_25M\_REF\_EN\_SHIFT) }
- PLL name.*
- enum `clock_pfd_t` {  
`kCLOCK_Pfd0` = 0U,  
`kCLOCK_Pfd1` = 1U,  
`kCLOCK_Pfd2` = 2U,  
`kCLOCK_Pfd3` = 3U }
- PLL PFD name.*
- enum `clock_output1_selection_t` {  
`kCLOCK_OutputPllUsb1Sw` = 0U,  
`kCLOCK_OutputPllSys` = 1U,  
`kCLOCK_OutputPllENET500M` = 2U,  
`kCLOCK_OutputSemcClk` = 5U,  
`kCLOCK_OutputAhbClk` = 0xBU,  
`kCLOCK_OutputIpgClk` = 0xCU,  
`kCLOCK_OutputPerClk` = 0xDU,  
`kCLOCK_OutputPll4MainClk` = 0xFU,  
`kCLOCK_DisableClockOutput1` = 0x10U }
- The enumerator of clock output1's clock source, such as USB1 PLL, SYS PLL and so on.*
- enum `clock_output2_selection_t` {

```

kCLOCK_OutputUsdhc1Clk = 3U,
kCLOCK_OutputLpi2cClk = 6U,
kCLOCK_OutputOscClk = 0xEU,
kCLOCK_OutputLpspiClk = 0x10U,
kCLOCK_OutputUsdhc2Clk = 0x11U,
kCLOCK_OutputSai1Clk = 0x12U,
kCLOCK_OutputSai2Clk = 0x13U,
kCLOCK_OutputSai3Clk = 0x14U,
kCLOCK_OutputTraceClk = 0x16U,
kCLOCK_OutputCanClk = 0x17U,
kCLOCK_OutputFlexspiClk = 0x1BU,
kCLOCK_OutputUartClk = 0x1CU,
kCLOCK_OutputSpdif0Clk = 0x1DU,
kCLOCK_DisableClockOutput2 = 0x1FU }

```

*The enumerator of clock output's clock source, such as USDHC1 clock root, LPI2C clock root and so on.*

- enum `clock_output_divider_t` {
 

```

kCLOCK_DivideBy1 = 0U,
kCLOCK_DivideBy2,
kCLOCK_DivideBy3,
kCLOCK_DivideBy4,
kCLOCK_DivideBy5,
kCLOCK_DivideBy6,
kCLOCK_DivideBy7,
kCLOCK_DivideBy8 }

```

*The enumerator of clock output's divider.*

- enum `clock_root_t` {
 

```

kCLOCK_Usdhc1ClkRoot = 0U,
kCLOCK_Usdhc2ClkRoot,
kCLOCK_FlexspiClkRoot,
kCLOCK_LpspiClkRoot,
kCLOCK_TraceClkRoot,
kCLOCK_Sai1ClkRoot,
kCLOCK_Sai2ClkRoot,
kCLOCK_Sai3ClkRoot,
kCLOCK_Lpi2cClkRoot,
kCLOCK_CanClkRoot,
kCLOCK_UartClkRoot,
kCLOCK_SpdifClkRoot,
kCLOCK_Flexio1ClkRoot }

```

*The enumerator of clock root.*

## Functions

- static void `CLOCK_SetMux` (`clock_mux_t` mux, `uint32_t` value)
 

*Set CCM MUX node to certain value.*
- static `uint32_t` `CLOCK_GetMux` (`clock_mux_t` mux)



- *Get CCM MUX value.*
- static void `CLOCK_SetDiv` (`clock_div_t` divider, `uint32_t` value)
  - *Set clock divider value.*
- static `uint32_t` `CLOCK_GetDiv` (`clock_div_t` divider)
  - *Get CCM DIV node value.*
- static void `CLOCK_ControlGate` (`clock_ip_name_t` name, `clock_gate_value_t` value)
  - *Control the clock gate for specific IP.*
- 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_SetMode` (`clock_mode_t` mode)
  - *Setting the low power mode that system will enter on next assertion of `dsm_request` signal.*
- static `uint32_t` `CLOCK_GetOscFreq` (void)
  - *Gets the OSC clock frequency.*
- `uint32_t` `CLOCK_GetAhbFreq` (void)
  - *Gets the AHB clock frequency.*
- `uint32_t` `CLOCK_GetSemcFreq` (void)
  - *Gets the SEMC clock frequency.*
- `uint32_t` `CLOCK_GetIpgFreq` (void)
  - *Gets the IPG clock frequency.*
- `uint32_t` `CLOCK_GetPerClkFreq` (void)
  - *Gets the PER clock frequency.*
- `uint32_t` `CLOCK_GetFreq` (`clock_name_t` name)
  - *Gets the clock frequency for a specific clock name.*
- static `uint32_t` `CLOCK_GetCpuClkFreq` (void)
  - *Get the CCM CPU/core/system frequency.*
- `uint32_t` `CLOCK_GetClockRootFreq` (`clock_root_t` clockRoot)
  - *Gets the frequency of selected clock root.*
- bool `CLOCK_EnableUsbhs0Clock` (`clock_usb_src_t` src, `uint32_t` freq)
  - *Enable USB HS clock.*

## Variables

- volatile `uint32_t` `g_xtalFreq`
  - *External XTAL (24M OSC/SYSOSC) clock frequency.*
- volatile `uint32_t` `g_rtcXtalFreq`
  - *External RTC XTAL (32K OSC) clock frequency.*

## Driver version

- #define `FSL_CLOCK_DRIVER_VERSION` (`MAKE_VERSION`(2, 5, 1))
  - *CLOCK driver version 2.5.1.*
- #define `SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY` (500000000UL)
- #define `CCM_ANALOG_PLL_BYPASS_SHIFT` (16U)
- #define `CCM_ANALOG_PLL_BYPASS_SHIFT` (16U)
- #define `CCM_ANALOG_PLL_BYPASS_CLK_SRC_MASK` (0xC000U)
- #define `CCM_ANALOG_PLL_BYPASS_CLK_SRC_MASK` (0xC000U)
- #define `CCM_ANALOG_PLL_BYPASS_CLK_SRC_SHIFT` (14U)
- #define `CCM_ANALOG_PLL_BYPASS_CLK_SRC_SHIFT` (14U)

## OSC operations

- void `CLOCK_InitExternalClk` (bool bypassXtalOsc)  
*Initialize the external 24MHz clock.*
- void `CLOCK_DeinitExternalClk` (void)  
*Deinitialize the external 24MHz clock.*
- void `CLOCK_SwitchOsc` (clock\_osc\_t osc)  
*Switch the OSC.*
- static uint32\_t `CLOCK_GetRtcFreq` (void)  
*Gets the RTC clock frequency.*
- static void `CLOCK_SetXtalFreq` (uint32\_t freq)  
*Set the XTAL (24M OSC) frequency based on board setting.*
- static void `CLOCK_SetRtcXtalFreq` (uint32\_t freq)  
*Set the RTC XTAL (32K OSC) frequency based on board setting.*
- void `CLOCK_InitRcOsc24M` (void)  
*Initialize the RC oscillator 24MHz clock.*
- void `CLOCK_DeinitRcOsc24M` (void)  
*Power down the RCOSC 24M clock.*

## 4.2 Data Structure Documentation

### 4.2.1 struct clock\_usb\_pll\_config\_t

#### Data Fields

- uint8\_t `loopDivider`  
*PLL loop divider.*
- uint8\_t `src`  
*Pll clock source, reference \_clock\_pll\_clk\_src.*

#### Field Documentation

##### (1) uint8\_t clock\_usb\_pll\_config\_t::loopDivider

0 - Fout=Fref\*20; 1 - Fout=Fref\*22

### 4.2.2 struct clock\_sys\_pll\_config\_t

#### Data Fields

- uint8\_t `loopDivider`  
*PLL loop divider.*
- uint32\_t `numerator`  
*30 bit numerator of fractional loop divider.*
- uint32\_t `denominator`  
*30 bit denominator of fractional loop divider*
- uint8\_t `src`  
*Pll clock source, reference \_clock\_pll\_clk\_src.*

- uint16\_t [ss\\_stop](#)  
*Stop value to get frequency change.*
- uint8\_t [ss\\_enable](#)  
*Enable spread spectrum modulation.*
- uint16\_t [ss\\_step](#)  
*Step value to get frequency change step.*

#### Field Documentation

##### (1) uint8\_t clock\_sys\_pll\_config\_t::loopDivider

Intended to be 1 (528M). 0 - Fout=Fref\*20; 1 - Fout=Fref\*22

##### (2) uint32\_t clock\_sys\_pll\_config\_t::numerator

##### (3) uint16\_t clock\_sys\_pll\_config\_t::ss\_stop

##### (4) uint16\_t clock\_sys\_pll\_config\_t::ss\_step

#### 4.2.3 struct clock\_audio\_pll\_config\_t

#### Data Fields

- uint8\_t [loopDivider](#)  
*PLL loop divider.*
- uint8\_t [postDivider](#)  
*Divider after the PLL, should only be 1, 2, 4, 8, 16.*
- uint32\_t [numerator](#)  
*30 bit numerator of fractional loop divider.*
- uint32\_t [denominator](#)  
*30 bit denominator of fractional loop divider*
- uint8\_t [src](#)  
*Pll clock source, reference \_clock\_pll\_clk\_src.*

#### Field Documentation

##### (1) uint8\_t clock\_audio\_pll\_config\_t::loopDivider

Valid range for DIV\_SELECT divider value: 27~54.

##### (2) uint8\_t clock\_audio\_pll\_config\_t::postDivider

##### (3) uint32\_t clock\_audio\_pll\_config\_t::numerator

#### 4.2.4 struct clock\_enet\_pll\_config\_t

#### Data Fields

- bool [enableClkOutput](#)

- *Power on and enable PLL clock output for ENET0 (ref\_enetpll0).*  
bool `enableClkOutput500M`
- *Power on and enable PLL clock output for ENET (ref\_enetpll500M).*  
bool `enableClkOutput25M`
- *Power on and enable PLL clock output for ENET1 (ref\_enetpll1).*  
uint8\_t `loopDivider`
- *Controls the frequency of the ENET0 reference clock.*  
uint8\_t `src`  
*Pll clock source, reference \_clock\_pll\_clk\_src.*

### Field Documentation

- (1) bool `clock_enet_pll_config_t::enableClkOutput`
- (2) bool `clock_enet_pll_config_t::enableClkOutput500M`
- (3) bool `clock_enet_pll_config_t::enableClkOutput25M`
- (4) uint8\_t `clock_enet_pll_config_t::loopDivider`

b00 25MHz b01 50MHz b10 100MHz (not 50% duty cycle) b11 125MHz

## 4.3 Macro Definition Documentation

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

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

Note

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

### 4.3.2 #define FSL\_CLOCK\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 1))

### 4.3.3 #define ADC\_CLOCKS

Value:

```
{
 kCLOCK_IpInvalid, kCLOCK_Adc1, kCLOCK_Adc2 \
}
```

#### 4.3.4 #define AOI\_CLOCKS

Value:

```
{
 kCLOCK_Aoi \
}
```

#### 4.3.5 #define BEE\_CLOCKS

Value:

```
{
 kCLOCK_Bee \
}
```

#### 4.3.6 #define CMP\_CLOCKS

Value:

```
{
 kCLOCK_IpInvalid, kCLOCK_Acmp1, kCLOCK_Acmp2, \
 kCLOCK_Acmp3, kCLOCK_Acmp4 \
}
```

#### 4.3.7 #define DCDC\_CLOCKS

Value:

```
{
 kCLOCK_Dcdc \
}
```

#### 4.3.8 #define DCP\_CLOCKS

Value:

```
{
 kCLOCK_Dcp \
}
```

### 4.3.9 #define DMAMUX\_CLOCKS

Value:

```
{
 kCLOCK_Dma \
```

### 4.3.10 #define EDMA\_CLOCKS

Value:

```
{
 kCLOCK_Dma \
```

### 4.3.11 #define ENC\_CLOCKS

Value:

```
{
 kCLOCK_IpInvalid, kCLOCK_Enc1, kCLOCK_Enc2 \
```

### 4.3.12 #define ENET\_CLOCKS

Value:

```
{
 kCLOCK_Enet \
```

### 4.3.13 #define EWM\_CLOCKS

Value:

```
{
 kCLOCK_Ewm0 \
```

#### 4.3.14 #define FLEXCAN\_CLOCKS

Value:

```
{
 kCLOCK_IpInvalid, kCLOCK_Can1, kCLOCK_Can2 \
}
```

#### 4.3.15 #define FLEXCAN\_PERIPH\_CLOCKS

Value:

```
{
 kCLOCK_IpInvalid, kCLOCK_Can1S, kCLOCK_Can2S \
}
```

#### 4.3.16 #define FLEXIO\_CLOCKS

Value:

```
{
 kCLOCK_IpInvalid, kCLOCK_Flexio1 \
}
```

#### 4.3.17 #define FLEXRAM\_CLOCKS

Value:

```
{
 kCLOCK_FlexRam \
}
```

#### 4.3.18 #define FLEXSPI\_CLOCKS

Value:

```
{
 kCLOCK_FlexSpi \
}
```

### 4.3.19 #define FLEXSPI\_EXSC\_CLOCKS

Value:

```
{
 kCLOCK_FlexSpiExsc \
}
```

### 4.3.20 #define GPIO\_CLOCKS

Value:

```
{
 kCLOCK_IpInvalid, kCLOCK_Gpio1, kCLOCK_Gpio2,
 kCLOCK_Gpio3, kCLOCK_IpInvalid, kCLOCK_Gpio5 \
}
```

### 4.3.21 #define GPT\_CLOCKS

Value:

```
{
 kCLOCK_IpInvalid, kCLOCK_Gpt1, kCLOCK_Gpt2 \
}
```

### 4.3.22 #define KPP\_CLOCKS

Value:

```
{
 kCLOCK_Kpp \
}
```

### 4.3.23 #define LPI2C\_CLOCKS

Value:

```
{
 kCLOCK_IpInvalid, kCLOCK_Lpi2c1, kCLOCK_Lpi2c2,
 kCLOCK_Lpi2c3, kCLOCK_Lpi2c4 \
}
```



#### 4.3.24 #define LPSPi\_CLOCKS

Value:

```
{
 kCLOCK_IpInvalid, kCLOCK_Lpspi1, kCLOCK_Lpspi2,
 kCLOCK_Lpspi3, kCLOCK_Lpspi4 \
}
```

#### 4.3.25 #define LPUART\_CLOCKS

Value:

```
{
 kCLOCK_IpInvalid, kCLOCK_Lpuart1, kCLOCK_Lpuart2,
 kCLOCK_Lpuart3, kCLOCK_Lpuart4, kCLOCK_Lpuart5,
 \
 kCLOCK_Lpuart6, kCLOCK_Lpuart7,
 kCLOCK_Lpuart8 \
}
```

#### 4.3.26 #define OCRAM\_EXSC\_CLOCKS

Value:

```
{
 kCLOCK_OcramExsc \
}
```

#### 4.3.27 #define PIT\_CLOCKS

Value:

```
{
 kCLOCK_Pit \
}
```

#### 4.3.28 #define PWM\_CLOCKS

Value:

```
{
 {kCLOCK_IpInvalid, kCLOCK_IpInvalid, kCLOCK_IpInvalid, kCLOCK_IpInvalid}, \
 {kCLOCK_Pwm1, kCLOCK_Pwm1, kCLOCK_Pwm1, kCLOCK_Pwm1}, \
 {
 \
 kCLOCK_Pwm2, kCLOCK_Pwm2, \
 kCLOCK_Pwm2, kCLOCK_Pwm2 \
 } \
}
```

### 4.3.29 #define RTWDOG\_CLOCKS

**Value:**

```
{
 \
 kCLOCK_Wdog3 \
}
```

### 4.3.30 #define SAI\_CLOCKS

**Value:**

```
{
 \
 kCLOCK_IpInvalid, kCLOCK_Sai1, kCLOCK_Sai2, \
 kCLOCK_Sai3 \
}
```

### 4.3.31 #define SEMC\_CLOCKS

**Value:**

```
{
 \
 kCLOCK_Semc \
}
```

### 4.3.32 #define SEMC\_EXSC\_CLOCKS

**Value:**

```
{
 \
 kCLOCK_SemcExsc \
}
```

### 4.3.33 #define TMR\_CLOCKS

Value:

```
{
 kCLOCK_IpInvalid, kCLOCK_Timer1, kCLOCK_Timer2 \
}
```

### 4.3.34 #define TRNG\_CLOCKS

Value:

```
{
 kCLOCK_Trng \
}
```

### 4.3.35 #define WDOG\_CLOCKS

Value:

```
{
 kCLOCK_IpInvalid, kCLOCK_Wdog1, kCLOCK_Wdog2 \
}
```

### 4.3.36 #define USDHC\_CLOCKS

Value:

```
{
 kCLOCK_IpInvalid, kCLOCK_Usdhc1, kCLOCK_Usdhc2 \
}
```

### 4.3.37 #define SPDIF\_CLOCKS

Value:

```
{
 kCLOCK_Spdif \
}
```

### 4.3.38 #define XBARA\_CLOCKS

Value:

```
{
 kCLOCK_Xbar1 \
}
```

### 4.3.39 #define XBARB\_CLOCKS

Value:

```
{
 kCLOCK_Xbar2 \
}
```

### 4.3.40 #define kCLOCK\_CoreSysClk kCLOCK\_CpuClk

### 4.3.41 #define CLOCK\_GetCoreSysClkFreq CLOCK\_GetCpuClkFreq

## 4.4 Enumeration Type Documentation

### 4.4.1 enum clock\_name\_t

Enumerator

*kCLOCK\_CpuClk* CPU clock.  
*kCLOCK\_AhbClk* AHB clock.  
*kCLOCK\_SemcClk* SEMC clock.  
*kCLOCK\_IpgClk* IPG clock.  
*kCLOCK\_PerClk* PER clock.  
*kCLOCK\_OscClk* OSC clock selected by PMU\_LOWPOWER\_CTRL[OSC\_SEL].  
*kCLOCK\_RtcClk* RTC clock. (RTCCLK)  
*kCLOCK\_Usb1PllClk* USB1PLLCLK.  
*kCLOCK\_Usb1Pllpfd0Clk* USB1PLLPDF0CLK.  
*kCLOCK\_Usb1Pllpfd1Clk* USB1PLLPDF1CLK.  
*kCLOCK\_Usb1Pllpfd2Clk* USB1PLLPDF2CLK.  
*kCLOCK\_Usb1Pllpfd3Clk* USB1PLLPDF3CLK.  
*kCLOCK\_Usb1SwClk* USB1PLLSWCLK.  
*kCLOCK\_Usb1Sw60MClk* USB1PLLSw60MCLK.  
*kCLOCK\_Usb1Sw80MClk* USB1PLLSw80MCLK.  
*kCLOCK\_SysPllClk* SYSPLLCLK.  
*kCLOCK\_SysPllpfd0Clk* SYSPLLPDF0CLK.

*kCLOCK\_SysPlIPfd1Clk* SYSPLLPFD1CLK.  
*kCLOCK\_SysPlIPfd2Clk* SYSPLLPFD2CLK.  
*kCLOCK\_SysPlIPfd3Clk* SYSPLLPFD3CLK.  
*kCLOCK\_EnetPlIClk* Enet PLLCLK ref\_enetpll.  
*kCLOCK\_EnetPlI25MClk* Enet PLLCLK ref\_enetpll25M.  
*kCLOCK\_EnetPlI500MClk* Enet PLLCLK ref\_enetpll500M.  
*kCLOCK\_AudioPlIClk* Audio PLLCLK.  
*kCLOCK\_NoneName* None Clock Name.

#### 4.4.2 enum clock\_ip\_name\_t

Enumerator

*kCLOCK\_Aips\_tz1* CCGR0, CG0.  
*kCLOCK\_Aips\_tz2* CCGR0, CG1.  
*kCLOCK\_Mqs* CCGR0, CG2.  
*kCLOCK\_Sim\_m\_clk\_r* CCGR0, CG4.  
*kCLOCK\_Dcp* CCGR0, CG5.  
*kCLOCK\_Lpuart3* CCGR0, CG6.  
*kCLOCK\_Can1* CCGR0, CG7.  
*kCLOCK\_Can1S* CCGR0, CG8.  
*kCLOCK\_Can2* CCGR0, CG9.  
*kCLOCK\_Can2S* CCGR0, CG10.  
*kCLOCK\_Trace* CCGR0, CG11.  
*kCLOCK\_Gpt2* CCGR0, CG12.  
*kCLOCK\_Gpt2S* CCGR0, CG13.  
*kCLOCK\_Lpuart2* CCGR0, CG14.  
*kCLOCK\_Gpio2* CCGR0, CG15.  
*kCLOCK\_Lpspi1* CCGR1, CG0.  
*kCLOCK\_Lpspi2* CCGR1, CG1.  
*kCLOCK\_Lpspi3* CCGR1, CG2.  
*kCLOCK\_Lpspi4* CCGR1, CG3.  
*kCLOCK\_Adc2* CCGR1, CG4.  
*kCLOCK\_Enet* CCGR1, CG5.  
*kCLOCK\_Pit* CCGR1, CG6.  
*kCLOCK\_Adc1* CCGR1, CG8.  
*kCLOCK\_SemcExsc* CCGR1, CG9.  
*kCLOCK\_Gpt1* CCGR1, CG10.  
*kCLOCK\_Gpt1S* CCGR1, CG11.  
*kCLOCK\_Lpuart4* CCGR1, CG12.  
*kCLOCK\_Gpio1* CCGR1, CG13.  
*kCLOCK\_Csu* CCGR1, CG14.  
*kCLOCK\_Gpio5* CCGR1, CG15.  
*kCLOCK\_OcramExsc* CCGR2, CG0.

*kCLOCK\_IomuxcSnvs* CCGR2, CG2.  
*kCLOCK\_Lpi2c1* CCGR2, CG3.  
*kCLOCK\_Lpi2c2* CCGR2, CG4.  
*kCLOCK\_Lpi2c3* CCGR2, CG5.  
*kCLOCK\_Ocotp* CCGR2, CG6.  
*kCLOCK\_Xbar1* CCGR2, CG11.  
*kCLOCK\_Xbar2* CCGR2, CG12.  
*kCLOCK\_Gpio3* CCGR2, CG13.  
*kCLOCK\_Lpuart5* CCGR3, CG1.  
*kCLOCK\_Semc* CCGR3, CG2.  
*kCLOCK\_Lpuart6* CCGR3, CG3.  
*kCLOCK\_Aoi* CCGR3, CG4.  
*kCLOCK\_Ewm0* CCGR3, CG7.  
*kCLOCK\_Wdog1* CCGR3, CG8.  
*kCLOCK\_FlexRam* CCGR3, CG9.  
*kCLOCK\_Acmp1* CCGR3, CG10.  
*kCLOCK\_Acmp2* CCGR3, CG11.  
*kCLOCK\_Acmp3* CCGR3, CG12.  
*kCLOCK\_Acmp4* CCGR3, CG13.  
*kCLOCK\_IomuxcSnvsGpr* CCGR3, CG15.  
*kCLOCK\_Sim\_m7\_clk\_r* CCGR4, CG0.  
*kCLOCK\_Iomuxc* CCGR4, CG1.  
*kCLOCK\_IomuxcGpr* CCGR4, CG2.  
*kCLOCK\_Bee* CCGR4, CG3.  
*kCLOCK\_SimM7* CCGR4, CG4.  
*kCLOCK\_SimM* CCGR4, CG6.  
*kCLOCK\_SimEms* CCGR4, CG7.  
*kCLOCK\_Pwm1* CCGR4, CG8.  
*kCLOCK\_Pwm2* CCGR4, CG9.  
*kCLOCK\_Enc1* CCGR4, CG12.  
*kCLOCK\_Enc2* CCGR4, CG13.  
*kCLOCK\_Rom* CCGR5, CG0.  
*kCLOCK\_Flexio1* CCGR5, CG1.  
*kCLOCK\_Wdog3* CCGR5, CG2.  
*kCLOCK\_Dma* CCGR5, CG3.  
*kCLOCK\_Kpp* CCGR5, CG4.  
*kCLOCK\_Wdog2* CCGR5, CG5.  
*kCLOCK\_Aips\_tz4* CCGR5, CG6.  
*kCLOCK\_Spdif* CCGR5, CG7.  
*kCLOCK\_Sai1* CCGR5, CG9.  
*kCLOCK\_Sai2* CCGR5, CG10.  
*kCLOCK\_Sai3* CCGR5, CG11.  
*kCLOCK\_Lpuart1* CCGR5, CG12.  
*kCLOCK\_Lpuart7* CCGR5, CG13.  
*kCLOCK\_SnvsHp* CCGR5, CG14.

*kCLOCK\_SnvsLp* CCGR5, CG15.  
*kCLOCK\_UsbOh3* CCGR6, CG0.  
*kCLOCK\_Usdhc1* CCGR6, CG1.  
*kCLOCK\_Usdhc2* CCGR6, CG2.  
*kCLOCK\_Dcdc* CCGR6, CG3.  
*kCLOCK\_Ipmux4* CCGR6, CG4.  
*kCLOCK\_FlexSpi* CCGR6, CG5.  
*kCLOCK\_Trng* CCGR6, CG6.  
*kCLOCK\_Lpuart8* CCGR6, CG7.  
*kCLOCK\_Timer4* CCGR6, CG8.  
*kCLOCK\_Aips\_tz3* CCGR6, CG9.  
*kCLOCK\_SimPer* CCGR6, CG10.  
*kCLOCK\_Anadig* CCGR6, CG11.  
*kCLOCK\_Lpi2c4* CCGR6, CG12.  
*kCLOCK\_Timer1* CCGR6, CG13.  
*kCLOCK\_Timer2* CCGR6, CG14.

#### 4.4.3 enum clock\_osc\_t

Enumerator

*kCLOCK\_RcOsc* On chip OSC.  
*kCLOCK\_XtalOsc* 24M Xtal OSC

#### 4.4.4 enum clock\_gate\_value\_t

Enumerator

*kCLOCK\_ClockNotNeeded* Clock is off during all modes.  
*kCLOCK\_ClockNeededRun* Clock is on in run mode, but off in WAIT and STOP modes.  
*kCLOCK\_ClockNeededRunWait* Clock is on during all modes, except STOP mode.

#### 4.4.5 enum clock\_mode\_t

Enumerator

*kCLOCK\_ModeRun* Remain in run mode.  
*kCLOCK\_ModeWait* Transfer to wait mode.  
*kCLOCK\_ModeStop* Transfer to stop mode.

#### 4.4.6 enum clock\_mux\_t

These constants define the mux control names for clock mux setting.

- 0:7: REG offset to CCM\_BASE in bytes.
- 8:15: Root clock setting bit field shift.
- 16:31: Root clock setting bit field width.

Enumerator

*kCLOCK\_Pll3SwMux* pll3\_sw\_clk mux name  
*kCLOCK\_PeriphMux* periph mux name  
*kCLOCK\_SemcAltMux* semc mux name  
*kCLOCK\_SemcMux* semc mux name  
*kCLOCK\_PrePeriphMux* pre-periph mux name  
*kCLOCK\_TraceMux* trace mux name  
*kCLOCK\_PeriphClk2Mux* periph clock2 mux name  
*kCLOCK\_LpspiMux* lpspi mux name  
*kCLOCK\_FlexspiMux* flexspi mux name  
*kCLOCK\_Usdhc2Mux* usdhc2 mux name  
*kCLOCK\_Usdhc1Mux* usdhc1 mux name  
*kCLOCK\_Sai3Mux* sai3 mux name  
*kCLOCK\_Sai2Mux* sai2 mux name  
*kCLOCK\_Sai1Mux* sai1 mux name  
*kCLOCK\_PerclkMux* perclk mux name  
*kCLOCK\_Flexio1Mux* flexio1 mux name  
*kCLOCK\_CanMux* can mux name  
*kCLOCK\_UartMux* uart mux name  
*kCLOCK\_SpdifMux* spdif mux name  
*kCLOCK\_Lpi2cMux* lpi2c mux name

#### 4.4.7 enum clock\_div\_t

These constants define div control names for clock div setting.

- 0:7: REG offset to CCM\_BASE in bytes.
- 8:15: Root clock setting bit field shift.
- 16:31: Root clock setting bit field width.

Enumerator

*kCLOCK\_ArmDiv* core div name  
*kCLOCK\_PeriphClk2Div* periph clock2 div name  
*kCLOCK\_SemcDiv* semc div name  
*kCLOCK\_AhbDiv* ahb div name



***kCLOCK\_IpgDiv*** ipg div name  
***kCLOCK\_LpspiDiv*** lpspi div name  
***kCLOCK\_FlexspiDiv*** flexspi div name  
***kCLOCK\_PerclkDiv*** perclk div name  
***kCLOCK\_CanDiv*** can div name  
***kCLOCK\_TraceDiv*** trace div name  
***kCLOCK\_Usdhc2Div*** usdhc2 div name  
***kCLOCK\_Usdhc1Div*** usdhc1 div name  
***kCLOCK\_UartDiv*** uart div name  
***kCLOCK\_Flexio1Div*** flexio1 pre div name  
***kCLOCK\_Sai3PreDiv*** sai3 pre div name  
***kCLOCK\_Sai3Div*** sai3 div name  
***kCLOCK\_Flexio1PreDiv*** flexio1 pre div name  
***kCLOCK\_Sai1PreDiv*** sai1 pre div name  
***kCLOCK\_Sai1Div*** sai1 div name  
***kCLOCK\_Sai2PreDiv*** sai2 pre div name  
***kCLOCK\_Sai2Div*** sai2 div name  
***kCLOCK\_Spdif0PreDiv*** spdif pre div name  
***kCLOCK\_Spdif0Div*** spdif div name  
***kCLOCK\_Lpi2cDiv*** lpi2c div name  
***kCLOCK\_NonePreDiv*** None Pre div.

#### 4.4.8 enum clock\_div\_value\_t

Enumerator

***kCLOCK\_ArmDivBy1*** ARM clock divider set to divided by 1.  
***kCLOCK\_ArmDivBy2*** ARM clock divider set to divided by 2.  
***kCLOCK\_ArmDivBy3*** ARM clock divider set to divided by 3.  
***kCLOCK\_ArmDivBy4*** ARM clock divider set to divided by 4.  
***kCLOCK\_ArmDivBy5*** ARM clock divider set to divided by 5.  
***kCLOCK\_ArmDivBy6*** ARM clock divider set to divided by 6.  
***kCLOCK\_ArmDivBy7*** ARM clock divider set to divided by 7.  
***kCLOCK\_ArmDivBy8*** ARM clock divider set to divided by 8.  
***kCLOCK\_PeriphClk2DivBy1*** PeriphClk2 divider set to divided by 1.  
***kCLOCK\_PeriphClk2DivBy2*** PeriphClk2 divider set to divided by 2.  
***kCLOCK\_PeriphClk2DivBy3*** PeriphClk2 divider set to divided by 3.  
***kCLOCK\_PeriphClk2DivBy4*** PeriphClk2 divider set to divided by 4.  
***kCLOCK\_PeriphClk2DivBy5*** PeriphClk2 divider set to divided by 5.  
***kCLOCK\_PeriphClk2DivBy6*** PeriphClk2 divider set to divided by 6.  
***kCLOCK\_PeriphClk2DivBy7*** PeriphClk2 divider set to divided by 7.  
***kCLOCK\_PeriphClk2DivBy8*** PeriphClk2 divider set to divided by 8.  
***kCLOCK\_SemcDivBy1*** SEMC divider set to divided by 1.  
***kCLOCK\_SemcDivBy2*** SEMC divider set to divided by 2.

***kCLOCK\_SemcDivBy3*** SEMC divider set to divided by 3.  
***kCLOCK\_SemcDivBy4*** SEMC divider set to divided by 4.  
***kCLOCK\_SemcDivBy5*** SEMC divider set to divided by 5.  
***kCLOCK\_SemcDivBy6*** SEMC divider set to divided by 6.  
***kCLOCK\_SemcDivBy7*** SEMC divider set to divided by 7.  
***kCLOCK\_SemcDivBy8*** SEMC divider set to divided by 8.  
***kCLOCK\_AhbDivBy1*** AHB divider set to divided by 1.  
***kCLOCK\_AhbDivBy2*** AHB divider set to divided by 2.  
***kCLOCK\_AhbDivBy3*** AHB divider set to divided by 3.  
***kCLOCK\_AhbDivBy4*** AHB divider set to divided by 4.  
***kCLOCK\_AhbDivBy5*** AHB divider set to divided by 5.  
***kCLOCK\_AhbDivBy6*** AHB divider set to divided by 6.  
***kCLOCK\_AhbDivBy7*** AHB divider set to divided by 7.  
***kCLOCK\_AhbDivBy8*** AHB divider set to divided by 8.  
***kCLOCK\_IpgDivBy1*** Ipg divider set to divided by 1.  
***kCLOCK\_IpgDivBy2*** Ipg divider set to divided by 2.  
***kCLOCK\_IpgDivBy3*** Ipg divider set to divided by 3.  
***kCLOCK\_IpgDivBy4*** Ipg divider set to divided by 4.  
***kCLOCK\_LpspiDivBy1*** LPSPI divider set to divided by 1.  
***kCLOCK\_LpspiDivBy2*** LPSPI divider set to divided by 2.  
***kCLOCK\_LpspiDivBy3*** LPSPI divider set to divided by 3.  
***kCLOCK\_LpspiDivBy4*** LPSPI divider set to divided by 4.  
***kCLOCK\_LpspiDivBy5*** LPSPI divider set to divided by 5.  
***kCLOCK\_LpspiDivBy6*** LPSPI divider set to divided by 6.  
***kCLOCK\_LpspiDivBy7*** LPSPI divider set to divided by 7.  
***kCLOCK\_LpspiDivBy8*** LPSPI divider set to divided by 8.  
***kCLOCK\_FlexspiDivBy1*** FLEXSPI divider set to divided by 1.  
***kCLOCK\_FlexspiDivBy2*** FLEXSPI divider set to divided by 2.  
***kCLOCK\_FlexspiDivBy3*** FLEXSPI divider set to divided by 3.  
***kCLOCK\_FlexspiDivBy4*** FLEXSPI divider set to divided by 4.  
***kCLOCK\_FlexspiDivBy5*** FLEXSPI divider set to divided by 5.  
***kCLOCK\_FlexspiDivBy6*** FLEXSPI divider set to divided by 6.  
***kCLOCK\_FlexspiDivBy7*** FLEXSPI divider set to divided by 7.  
***kCLOCK\_FlexspiDivBy8*** FLEXSPI divider set to divided by 8.  
***kCLOCK\_TraceDivBy1*** TRACE divider set to divided by 1.  
***kCLOCK\_TraceDivBy2*** TRACE divider set to divided by 2.  
***kCLOCK\_TraceDivBy3*** TRACE divider set to divided by 3.  
***kCLOCK\_TraceDivBy4*** TRACE divider set to divided by 4.  
***kCLOCK\_Usdhc2DivBy1*** USDHC2 divider set to divided by 1.  
***kCLOCK\_Usdhc2DivBy2*** USDHC2 divider set to divided by 2.  
***kCLOCK\_Usdhc2DivBy3*** USDHC2 divider set to divided by 3.  
***kCLOCK\_Usdhc2DivBy4*** USDHC2 divider set to divided by 4.  
***kCLOCK\_Usdhc2DivBy5*** USDHC2 divider set to divided by 5.  
***kCLOCK\_Usdhc2DivBy6*** USDHC2 divider set to divided by 6.  
***kCLOCK\_Usdhc2DivBy7*** USDHC2 divider set to divided by 7.

|                                |                                           |
|--------------------------------|-------------------------------------------|
| <i>kCLOCK_Usdhc2DivBy8</i>     | USDHC2 divider set to divided by 8.       |
| <i>kCLOCK_Usdhc1DivBy1</i>     | USDHC1 divider set to divided by 1.       |
| <i>kCLOCK_Usdhc1DivBy2</i>     | USDHC1 divider set to divided by 2.       |
| <i>kCLOCK_Usdhc1DivBy3</i>     | USDHC1 divider set to divided by 3.       |
| <i>kCLOCK_Usdhc1DivBy4</i>     | USDHC1 divider set to divided by 4.       |
| <i>kCLOCK_Usdhc1DivBy5</i>     | USDHC1 divider set to divided by 5.       |
| <i>kCLOCK_Usdhc1DivBy6</i>     | USDHC1 divider set to divided by 6.       |
| <i>kCLOCK_Usdhc1DivBy7</i>     | USDHC1 divider set to divided by 7.       |
| <i>kCLOCK_Usdhc1DivBy8</i>     | USDHC1 divider set to divided by 8.       |
| <i>kCLOCK_Flexio1DivBy1</i>    | Flexio1 divider set to divided by 1.      |
| <i>kCLOCK_Flexio1DivBy2</i>    | Flexio1 divider set to divided by 2.      |
| <i>kCLOCK_Flexio1DivBy3</i>    | Flexio1 divider set to divided by 3.      |
| <i>kCLOCK_Flexio1DivBy4</i>    | Flexio1 divider set to divided by 4.      |
| <i>kCLOCK_Flexio1DivBy5</i>    | Flexio1 divider set to divided by 5.      |
| <i>kCLOCK_Flexio1DivBy6</i>    | Flexio1 divider set to divided by 6.      |
| <i>kCLOCK_Flexio1DivBy7</i>    | Flexio1 divider set to divided by 7.      |
| <i>kCLOCK_Flexio1DivBy8</i>    | Flexio1 divider set to divided by 8.      |
| <i>kCLOCK_Sai3PreDivBy1</i>    | SAI3ClkPred divider set to divided by 1.  |
| <i>kCLOCK_Sai3PreDivBy2</i>    | SAI3ClkPred divider set to divided by 2.  |
| <i>kCLOCK_Sai3PreDivBy3</i>    | SAI3ClkPred divider set to divided by 3.  |
| <i>kCLOCK_Sai3PreDivBy4</i>    | SAI3ClkPred divider set to divided by 4.  |
| <i>kCLOCK_Sai3PreDivBy5</i>    | SAI3ClkPred divider set to divided by 5.  |
| <i>kCLOCK_Sai3PreDivBy6</i>    | SAI3ClkPred divider set to divided by 6.  |
| <i>kCLOCK_Sai3PreDivBy7</i>    | SAI3ClkPred divider set to divided by 7.  |
| <i>kCLOCK_Sai3PreDivBy8</i>    | SAI3ClkPred divider set to divided by 8.  |
| <i>kCLOCK_Flexio1PreDivBy1</i> | Flexio1 pred divider set to divided by 1. |
| <i>kCLOCK_Flexio1PreDivBy2</i> | Flexio1 pred divider set to divided by 2. |
| <i>kCLOCK_Flexio1PreDivBy3</i> | Flexio1 pred divider set to divided by 3. |
| <i>kCLOCK_Flexio1PreDivBy4</i> | Flexio1 pred divider set to divided by 4. |
| <i>kCLOCK_Flexio1PreDivBy5</i> | Flexio1 pred divider set to divided by 5. |
| <i>kCLOCK_Flexio1PreDivBy6</i> | Flexio1 pred divider set to divided by 6. |
| <i>kCLOCK_Flexio1PreDivBy7</i> | Flexio1 pred divider set to divided by 7. |
| <i>kCLOCK_Flexio1PreDivBy8</i> | Flexio1 pred divider set to divided by 8. |
| <i>kCLOCK_Sai1PreDivBy1</i>    | SAI1 pred divider set to divided by 1.    |
| <i>kCLOCK_Sai1PreDivBy2</i>    | SAI1 pred divider set to divided by 2.    |
| <i>kCLOCK_Sai1PreDivBy3</i>    | SAI1 pred divider set to divided by 3.    |
| <i>kCLOCK_Sai1PreDivBy4</i>    | SAI1 pred divider set to divided by 4.    |
| <i>kCLOCK_Sai1PreDivBy5</i>    | SAI1 pred divider set to divided by 5.    |
| <i>kCLOCK_Sai1PreDivBy6</i>    | SAI1 pred divider set to divided by 6.    |
| <i>kCLOCK_Sai1PreDivBy7</i>    | SAI1 pred divider set to divided by 7.    |
| <i>kCLOCK_Sai1PreDivBy8</i>    | SAI1 pred divider set to divided by 8.    |
| <i>kCLOCK_Sai2PreDivBy1</i>    | SAI2ClkPred divider set to divided by 1.  |
| <i>kCLOCK_Sai2PreDivBy2</i>    | SAI2ClkPred divider set to divided by 2.  |
| <i>kCLOCK_Sai2PreDivBy3</i>    | SAI2ClkPred divider set to divided by 3.  |
| <i>kCLOCK_Sai2PreDivBy4</i>    | SAI2ClkPred divider set to divided by 4.  |

- kCLOCK\_Sai2PreDivBy5* SAI2ClkPred divider set to divided by 5.
- kCLOCK\_Sai2PreDivBy6* SAI2ClkPred divider set to divided by 6.
- kCLOCK\_Sai2PreDivBy7* SAI2ClkPred divider set to divided by 7.
- kCLOCK\_Sai2PreDivBy8* SAI2ClkPred divider set to divided by 8.
- kCLOCK\_Spdif0PreDivBy1* SPDIF0ClkPred divider set to divided by 1.
- kCLOCK\_Spdif0PreDivBy2* SPDIF0ClkPred divider set to divided by 2.
- kCLOCK\_Spdif0PreDivBy3* SPDIF0ClkPred divider set to divided by 3.
- kCLOCK\_Spdif0PreDivBy4* SPDIF0ClkPred divider set to divided by 4.
- kCLOCK\_Spdif0PreDivBy5* SPDIF0ClkPred divider set to divided by 5.
- kCLOCK\_Spdif0PreDivBy6* SPDIF0ClkPred divider set to divided by 6.
- kCLOCK\_Spdif0PreDivBy7* SPDIF0ClkPred divider set to divided by 7.
- kCLOCK\_Spdif0PreDivBy8* SPDIF0ClkPred divider set to divided by 8.
- kCLOCK\_Spdif0DivBy1* SPDIF0ClkPodf divider set to divided by 1.
- kCLOCK\_Spdif0DivBy2* SPDIF0ClkPodf divider set to divided by 2.
- kCLOCK\_Spdif0DivBy3* SPDIF0ClkPodf divider set to divided by 3.
- kCLOCK\_Spdif0DivBy4* SPDIF0ClkPodf divider set to divided by 4.
- kCLOCK\_Spdif0DivBy5* SPDIF0ClkPodf divider set to divided by 5.
- kCLOCK\_Spdif0DivBy6* SPDIF0ClkPodf divider set to divided by 6.
- kCLOCK\_Spdif0DivBy7* SPDIF0ClkPodf divider set to divided by 7.
- kCLOCK\_Spdif0DivBy8* SPDIF0ClkPodf divider set to divided by 8.
- kCLOCK\_MiscDivBy1* Misc divider like LPI2C set to divided by 1.
- kCLOCK\_MiscDivBy2* Misc divider like LPI2C set to divided by 2.
- kCLOCK\_MiscDivBy3* Misc divider like LPI2C set to divided by 3.
- kCLOCK\_MiscDivBy4* Misc divider like LPI2C set to divided by 4.
- kCLOCK\_MiscDivBy5* Misc divider like LPI2C set to divided by 5.
- kCLOCK\_MiscDivBy6* Misc divider like LPI2C set to divided by 6.
- kCLOCK\_MiscDivBy7* Misc divider like LPI2C set to divided by 7.
- kCLOCK\_MiscDivBy8* Misc divider like LPI2C set to divided by 8.
- kCLOCK\_MiscDivBy9* Misc divider like LPI2C set to divided by 9.
- kCLOCK\_MiscDivBy10* Misc divider like LPI2C set to divided by 10.
- kCLOCK\_MiscDivBy11* Misc divider like LPI2C set to divided by 11.
- kCLOCK\_MiscDivBy12* Misc divider like LPI2C set to divided by 12.
- kCLOCK\_MiscDivBy13* Misc divider like LPI2C set to divided by 13.
- kCLOCK\_MiscDivBy14* Misc divider like LPI2C set to divided by 14.
- kCLOCK\_MiscDivBy15* Misc divider like LPI2C set to divided by 15.
- kCLOCK\_MiscDivBy16* Misc divider like LPI2C set to divided by 16.
- kCLOCK\_MiscDivBy17* Misc divider like LPI2C set to divided by 17.
- kCLOCK\_MiscDivBy18* Misc divider like LPI2C set to divided by 18.
- kCLOCK\_MiscDivBy19* Misc divider like LPI2C set to divided by 19.
- kCLOCK\_MiscDivBy20* Misc divider like LPI2C set to divided by 20.
- kCLOCK\_MiscDivBy21* Misc divider like LPI2C set to divided by 21.
- kCLOCK\_MiscDivBy22* Misc divider like LPI2C set to divided by 22.
- kCLOCK\_MiscDivBy23* Misc divider like LPI2C set to divided by 23.
- kCLOCK\_MiscDivBy24* Misc divider like LPI2C set to divided by 24.
- kCLOCK\_MiscDivBy25* Misc divider like LPI2C set to divided by 25.

|                                  |                                               |
|----------------------------------|-----------------------------------------------|
| <b><i>kCLOCK_MiscDivBy26</i></b> | Misc divider like LPI2C set to divided by 26. |
| <b><i>kCLOCK_MiscDivBy27</i></b> | Misc divider like LPI2C set to divided by 27. |
| <b><i>kCLOCK_MiscDivBy28</i></b> | Misc divider like LPI2C set to divided by 28. |
| <b><i>kCLOCK_MiscDivBy29</i></b> | Misc divider like LPI2C set to divided by 29. |
| <b><i>kCLOCK_MiscDivBy30</i></b> | Misc divider like LPI2C set to divided by 30. |
| <b><i>kCLOCK_MiscDivBy31</i></b> | Misc divider like LPI2C set to divided by 31. |
| <b><i>kCLOCK_MiscDivBy32</i></b> | Misc divider like LPI2C set to divided by 32. |
| <b><i>kCLOCK_MiscDivBy33</i></b> | Misc divider like LPI2C set to divided by 33. |
| <b><i>kCLOCK_MiscDivBy34</i></b> | Misc divider like LPI2C set to divided by 34. |
| <b><i>kCLOCK_MiscDivBy35</i></b> | Misc divider like LPI2C set to divided by 35. |
| <b><i>kCLOCK_MiscDivBy36</i></b> | Misc divider like LPI2C set to divided by 36. |
| <b><i>kCLOCK_MiscDivBy37</i></b> | Misc divider like LPI2C set to divided by 37. |
| <b><i>kCLOCK_MiscDivBy38</i></b> | Misc divider like LPI2C set to divided by 38. |
| <b><i>kCLOCK_MiscDivBy39</i></b> | Misc divider like LPI2C set to divided by 39. |
| <b><i>kCLOCK_MiscDivBy40</i></b> | Misc divider like LPI2C set to divided by 40. |
| <b><i>kCLOCK_MiscDivBy41</i></b> | Misc divider like LPI2C set to divided by 41. |
| <b><i>kCLOCK_MiscDivBy42</i></b> | Misc divider like LPI2C set to divided by 42. |
| <b><i>kCLOCK_MiscDivBy43</i></b> | Misc divider like LPI2C set to divided by 43. |
| <b><i>kCLOCK_MiscDivBy44</i></b> | Misc divider like LPI2C set to divided by 44. |
| <b><i>kCLOCK_MiscDivBy45</i></b> | Misc divider like LPI2C set to divided by 45. |
| <b><i>kCLOCK_MiscDivBy46</i></b> | Misc divider like LPI2C set to divided by 46. |
| <b><i>kCLOCK_MiscDivBy47</i></b> | Misc divider like LPI2C set to divided by 47. |
| <b><i>kCLOCK_MiscDivBy48</i></b> | Misc divider like LPI2C set to divided by 48. |
| <b><i>kCLOCK_MiscDivBy49</i></b> | Misc divider like LPI2C set to divided by 49. |
| <b><i>kCLOCK_MiscDivBy50</i></b> | Misc divider like LPI2C set to divided by 50. |
| <b><i>kCLOCK_MiscDivBy51</i></b> | Misc divider like LPI2C set to divided by 51. |
| <b><i>kCLOCK_MiscDivBy52</i></b> | Misc divider like LPI2C set to divided by 52. |
| <b><i>kCLOCK_MiscDivBy53</i></b> | Misc divider like LPI2C set to divided by 53. |
| <b><i>kCLOCK_MiscDivBy54</i></b> | Misc divider like LPI2C set to divided by 54. |
| <b><i>kCLOCK_MiscDivBy55</i></b> | Misc divider like LPI2C set to divided by 55. |
| <b><i>kCLOCK_MiscDivBy56</i></b> | Misc divider like LPI2C set to divided by 56. |
| <b><i>kCLOCK_MiscDivBy57</i></b> | Misc divider like LPI2C set to divided by 57. |
| <b><i>kCLOCK_MiscDivBy58</i></b> | Misc divider like LPI2C set to divided by 58. |
| <b><i>kCLOCK_MiscDivBy59</i></b> | Misc divider like LPI2C set to divided by 59. |
| <b><i>kCLOCK_MiscDivBy60</i></b> | Misc divider like LPI2C set to divided by 60. |
| <b><i>kCLOCK_MiscDivBy61</i></b> | Misc divider like LPI2C set to divided by 61. |
| <b><i>kCLOCK_MiscDivBy62</i></b> | Misc divider like LPI2C set to divided by 62. |
| <b><i>kCLOCK_MiscDivBy63</i></b> | Misc divider like LPI2C set to divided by 63. |
| <b><i>kCLOCK_MiscDivBy64</i></b> | Misc divider like LPI2C set to divided by 64. |

#### 4.4.9 enum clock\_usb\_src\_t

Enumerator

*kCLOCK\_Usb480M* Use 480M.

*kCLOCK\_UsbSrcUnused* Used when the function does not care the clock source.

#### 4.4.10 enum clock\_usb\_phy\_src\_t

Enumerator

*kCLOCK\_Usbphy480M* Use 480M.

#### 4.4.11 enum \_clock\_pll\_clk\_src

Enumerator

*kCLOCK\_PllClkSrc24M* Pll clock source 24M.

*kCLOCK\_PllSrcClkPN* Pll clock source CLK1\_P and CLK1\_N.

#### 4.4.12 enum clock\_pll\_t

Enumerator

*kCLOCK\_PllSys* PLL SYS.

*kCLOCK\_PllUsb1* PLL USB1.

*kCLOCK\_PllAudio* PLL Audio.

*kCLOCK\_PllEnet* PLL Enet0.

*kCLOCK\_PllEnet500M* PLL ENET.

*kCLOCK\_PllEnet25M* PLL Enet1.

#### 4.4.13 enum clock\_pfd\_t

Enumerator

*kCLOCK\_Pfd0* PLL PFD0.

*kCLOCK\_Pfd1* PLL PFD1.

*kCLOCK\_Pfd2* PLL PFD2.

*kCLOCK\_Pfd3* PLL PFD3.

#### 4.4.14 enum clock\_output1\_selection\_t

Enumerator

*kCLOCK\_OutputPllUsb1Sw* Selects USB1 PLL SW clock(Divided by 2) output.  
*kCLOCK\_OutputPllSys* Selects SYS PLL clock(Divided by 2) output.  
*kCLOCK\_OutputPllENET500M* Selects ENET PLL clock(Divided by 2) output.  
*kCLOCK\_OutputSemcClk* Selects semc clock root output.  
*kCLOCK\_OutputAhbClk* Selects AHB clock root output.  
*kCLOCK\_OutputIpgClk* Selects IPG clock root output.  
*kCLOCK\_OutputPerClk* Selects PERCLK clock root output.  
*kCLOCK\_OutputPll4MainClk* Selects PLL4 main clock output.  
*kCLOCK\_DisableClockOutput1* Disables CLK01.

#### 4.4.15 enum clock\_output2\_selection\_t

Enumerator

*kCLOCK\_OutputUsdhc1Clk* Selects USDHC1 clock root output.  
*kCLOCK\_OutputLpi2cClk* Selects LPI2C clock root output.  
*kCLOCK\_OutputOscClk* Selects OSC output.  
*kCLOCK\_OutputLpspiClk* Selects LPSPI clock root output.  
*kCLOCK\_OutputUsdhc2Clk* Selects USDHC2 clock root output.  
*kCLOCK\_OutputSai1Clk* Selects SAI1 clock root output.  
*kCLOCK\_OutputSai2Clk* Selects SAI2 clock root output.  
*kCLOCK\_OutputSai3Clk* Selects SAI3 clock root output.  
*kCLOCK\_OutputTraceClk* Selects Trace clock root output.  
*kCLOCK\_OutputCanClk* Selects CAN clock root output.  
*kCLOCK\_OutputFlexspiClk* Selects FLEXSPI clock root output.  
*kCLOCK\_OutputUartClk* Selects UART clock root output.  
*kCLOCK\_OutputSpdif0Clk* Selects SPDIF0 clock root output.  
*kCLOCK\_DisableClockOutput2* Disables CLK02.

#### 4.4.16 enum clock\_output\_divider\_t

Enumerator

*kCLOCK\_DivideBy1* Output clock divided by 1.  
*kCLOCK\_DivideBy2* Output clock divided by 2.  
*kCLOCK\_DivideBy3* Output clock divided by 3.  
*kCLOCK\_DivideBy4* Output clock divided by 4.  
*kCLOCK\_DivideBy5* Output clock divided by 5.  
*kCLOCK\_DivideBy6* Output clock divided by 6.

*kCLOCK\_DivideBy7* Output clock divided by 7.

*kCLOCK\_DivideBy8* Output clock divided by 8.

#### 4.4.17 enum clock\_root\_t

Enumerator

*kCLOCK\_Usdhc1ClkRoot* USDHC1 clock root.

*kCLOCK\_Usdhc2ClkRoot* USDHC2 clock root.

*kCLOCK\_FlexspiClkRoot* FLEXSPI clock root.

*kCLOCK\_LpspiClkRoot* LPSPI clock root.

*kCLOCK\_TraceClkRoot* Trace clock root.

*kCLOCK\_Sai1ClkRoot* SAI1 clock root.

*kCLOCK\_Sai2ClkRoot* SAI2 clock root.

*kCLOCK\_Sai3ClkRoot* SAI3 clock root.

*kCLOCK\_Lpi2cClkRoot* LPI2C clock root.

*kCLOCK\_CanClkRoot* CAN clock root.

*kCLOCK\_UartClkRoot* UART clock root.

*kCLOCK\_SpdifClkRoot* SPDIF clock root.

*kCLOCK\_Flexio1ClkRoot* FLEXIO1 clock root.

## 4.5 Function Documentation

### 4.5.1 static void CLOCK\_SetMux ( clock\_mux\_t mux, uint32\_t value ) [inline], [static]

Parameters

|              |                                                                  |
|--------------|------------------------------------------------------------------|
| <i>mux</i>   | Which mux node to set, see <a href="#">clock_mux_t</a> .         |
| <i>value</i> | Clock mux value to set, different mux has different value range. |

### 4.5.2 static uint32\_t CLOCK\_GetMux ( clock\_mux\_t mux ) [inline], [static]

Parameters

|            |                                                          |
|------------|----------------------------------------------------------|
| <i>mux</i> | Which mux node to get, see <a href="#">clock_mux_t</a> . |
|------------|----------------------------------------------------------|

Returns

Clock mux value.



### 4.5.3 static void CLOCK\_SetDiv ( clock\_div\_t *divider*, uint32\_t *value* ) [inline], [static]

Example, set the ARM clock divider to divide by 2:

```
CLOCK_SetDiv(kCLOCK_ArmDiv, kCLOCK_ArmDivBy2);
```

Example, set the LPI2C clock divider to divide by 5.

```
CLOCK_SetDiv(kCLOCK_Lpi2cDiv, kCLOCK_MiscDivBy5);
```

Only [kCLOCK\\_PerclkDiv](#), [kCLOCK\\_CanDiv](#), [kCLOCK\\_UartDiv](#), [kCLOCK\\_Sai3Div](#), [kCLOCK\\_Sai1-Div](#), [kCLOCK\\_Sai2Div](#), [kCLOCK\\_Lpi2cDiv](#) can use the divider [kCLOCK\\_MiscDivByxxx](#).

Parameters

|                |                                                                                                                                                                                           |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>divider</i> | Which divider node to set.                                                                                                                                                                |
| <i>value</i>   | Clock div value to set, different divider has different value range. See <a href="#">clock_div_value_t</a> for details. Divided clock frequency = Undivided clock frequency / (value + 1) |

### 4.5.4 static uint32\_t CLOCK\_GetDiv ( clock\_div\_t *divider* ) [inline], [static]

Parameters

|                |                                                          |
|----------------|----------------------------------------------------------|
| <i>divider</i> | Which div node to get, see <a href="#">clock_div_t</a> . |
|----------------|----------------------------------------------------------|

### 4.5.5 static void CLOCK\_ControlGate ( clock\_ip\_name\_t *name*, clock\_gate\_value\_t *value* ) [inline], [static]

Parameters

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

|              |                                                                   |
|--------------|-------------------------------------------------------------------|
| <i>value</i> | Clock gate value to set, see <a href="#">clock_gate_value_t</a> . |
|--------------|-------------------------------------------------------------------|

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

Parameters

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

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

Parameters

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

#### 4.5.8 **static void CLOCK\_SetMode ( clock\_mode\_t mode ) [inline], [static]**

Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>mode</i> | Which mode to enter, see <a href="#">clock_mode_t</a> . |
|-------------|---------------------------------------------------------|

#### 4.5.9 **static uint32\_t CLOCK\_GetOscFreq ( void ) [inline], [static]**

This function will return the external XTAL OSC frequency if it is selected as the source of OSC, otherwise internal 24MHz RC OSC frequency will be returned.

Returns

Clock frequency; If the clock is invalid, returns 0.

#### 4.5.10 **uint32\_t CLOCK\_GetAhbFreq ( void )**

Returns

The AHB clock frequency value in hertz.

**4.5.11 uint32\_t CLOCK\_GetSemcFreq ( void )**

Returns

The SEMC clock frequency value in hertz.

**4.5.12 uint32\_t CLOCK\_GetIpgFreq ( void )**

Returns

The IPG clock frequency value in hertz.

**4.5.13 uint32\_t CLOCK\_GetPerClkFreq ( void )**

Returns

The PER clock frequency value in hertz.

**4.5.14 uint32\_t CLOCK\_GetFreq ( clock\_name\_t *name* )**

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in `clock_name_t`.

Parameters

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

Returns

Clock frequency value in hertz

**4.5.15 static uint32\_t CLOCK\_GetCpuClkFreq ( void ) [inline], [static]**

Returns

Clock frequency; If the clock is invalid, returns 0.

**4.5.16 uint32\_t CLOCK\_GetClockRootFreq ( clock\_root\_t *clockRoot* )**

## Parameters

|                  |                                                                                          |
|------------------|------------------------------------------------------------------------------------------|
| <i>clockRoot</i> | The clock root used to get the frequency, please refer to <a href="#">clock_root_t</a> . |
|------------------|------------------------------------------------------------------------------------------|

## Returns

The frequency of selected clock root.

#### 4.5.17 void CLOCK\_InitExternalClk ( bool *bypassXtalOsc* )

This function supports two modes:

1. Use external crystal oscillator.
2. Bypass the external crystal oscillator, using input source clock directly.

After this function, please call `CLOCK_SetXtal0Freq` to inform clock driver the external clock frequency.

## Parameters

|                      |                                                         |
|----------------------|---------------------------------------------------------|
| <i>bypassXtalOsc</i> | Pass in true to bypass the external crystal oscillator. |
|----------------------|---------------------------------------------------------|

## Note

This device does not support bypass external crystal oscillator, so the input parameter should always be false.

#### 4.5.18 void CLOCK\_DeinitExternalClk ( void )

This function disables the external 24MHz clock.

After this function, please call `CLOCK_SetXtal0Freq` to set external clock frequency to 0.

#### 4.5.19 void CLOCK\_SwitchOsc ( clock\_osc\_t *osc* )

This function switches the OSC source for SoC.

## Parameters

|            |                          |
|------------|--------------------------|
| <i>osc</i> | OSC source to switch to. |
|------------|--------------------------|

**4.5.20 static uint32\_t CLOCK\_GetRtcFreq ( void ) [inline], [static]**

Returns

Clock frequency; If the clock is invalid, returns 0.

**4.5.21 static void CLOCK\_SetXtalFreq ( uint32\_t freq ) [inline], [static]**

Parameters

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

**4.5.22 static void CLOCK\_SetRtcXtalFreq ( uint32\_t freq ) [inline], [static]**

Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>freq</i> | The RTC XTAL input clock frequency in Hz. |
|-------------|-------------------------------------------|

**4.5.23 bool CLOCK\_EnableUsbhs0Clock ( clock\_usb\_src\_t src, uint32\_t freq )**

This function only enables the access to USB HS peripheral, upper layer should first call the `CLOCK_EnableUsbhs0PhyPllClock` to enable the PHY clock to use USB HS.

Parameters

|             |                                                                                             |
|-------------|---------------------------------------------------------------------------------------------|
| <i>src</i>  | USB HS does not care about the clock source, here must be <code>kCLOCK_UsbSrc_Used</code> . |
| <i>freq</i> | USB HS does not care about the clock source, so this parameter is ignored.                  |

Return values

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

**4.6 Variable Documentation****4.6.1 volatile uint32\_t g\_xtalFreq**

The XTAL (24M OSC/SYSOSC) clock frequency in Hz, when the clock is setup, use the function `CLOCK_SetXtalFreq` to set the value in to clock driver. For example, if XTAL is 24MHz,

```
* CLOCK_InitExternalClk(false);
* CLOCK_SetXtalFreq(24000000);
*
```

#### 4.6.2 volatile uint32\_t g\_rtcXtalFreq

The RTC XTAL (32K OSC) clock frequency in Hz, when the clock is setup, use the function `CLOCK_SetRtcXtalFreq` to set the value in to clock driver.

# Chapter 5

## ROMAPI Driver

### 5.1 Overview

The ROMAPI driver provides the functionalities to operate the external NOR Flash connected to the FLEXSPI controller.

The ROMAPI driver supports:

- Initialize serial NOR flash via FLEXSPI
- Program data to serial NOR flash via FLEXSPI.
- Erase serial NOR flash region via FLEXSPI.

### Data Structures

- struct `flexspi_lut_seq_t`  
*FLEXSPI LUT Sequence structure. [More...](#)*
- struct `flexspi_mem_config_t`  
*FLEXSPI Memory Configuration Block. [More...](#)*
- struct `flexspi_nor_config_t`  
*Serial NOR configuration block. [More...](#)*
- struct `flexspi_xfer_t`  
*FLEXSPI Transfer Context. [More...](#)*

### Macros

- #define `FSL_ROM_ROMAPI_VERSION` (MAKE\_VERSION(1U, 1U, 0U))  
*ROMAPI version 1.1.0.*
- #define `FSL_ROM_FLEXSPINOR_DRIVER_VERSION` (MAKE\_VERSION(1U, 4U, 0U))  
*ROM FLEXSPI NOR driver version 1.4.0.*
- #define `kROM_StatusGroup_FLEXSPI` 60U  
*ROM FLEXSPI status group number.*
- #define `kROM_StatusGroup_FLEXSPINOR` 200U  
*ROM FLEXSPI NOR status group number.*
- #define `FSL_ROM_FLEXSPI_BITMASK`(bit\_offset) (1U << (bit\_offset))  
*Generate bit mask.*
- #define `FLEXSPI_CFG_BLK_TAG` (0x42464346UL)  
*FLEXSPI memory config block related definitions.*
- #define `FLEXSPI_CFG_BLK_VERSION` (0x56010400UL)  
*V1.4.0.*
- #define `NOR_CMD_LUT_SEQ_IDX_READ` 0U  
*NOR LUT sequence index used for default LUT assignment NOTE: The will take effect if the lut sequences are not customized.*
- #define `NOR_CMD_LUT_SEQ_IDX_READSTATUS` 1U  
*Read Status LUT sequence id in lookupTable stored in config block.*
- #define `NOR_CMD_LUT_SEQ_IDX_READSTATUS_XPI` 2U

- *Read status DPI/QPI/OPI sequence id in lookupTable stored in config block.*
- #define `NOR_CMD_LUT_SEQ_IDX_WRITEENABLE` 3U  
*Write Enable sequence id in lookupTable stored in config block.*
- #define `NOR_CMD_LUT_SEQ_IDX_WRITEENABLE_XPI` 4U  
*Write Enable DPI/QPI/OPI sequence id in lookupTable stored in config block.*
- #define `NOR_CMD_LUT_SEQ_IDX_ERASESECTOR` 5U  
*Erase Sector sequence id in lookupTable stored in config block.*
- #define `NOR_CMD_LUT_SEQ_IDX_ERASEBLOCK` 8U  
*Erase Block sequence id in lookupTable stored in config block.*
- #define `NOR_CMD_LUT_SEQ_IDX_PAGEPROGRAM` 9U  
*Program sequence id in lookupTable stored in config block.*
- #define `NOR_CMD_LUT_SEQ_IDX_CHIPERASE` 11U  
*Chip Erase sequence in lookupTable id stored in config block.*
- #define `NOR_CMD_LUT_SEQ_IDX_READ_SFDP` 13U  
*Read SFDP sequence in lookupTable id stored in config block.*
- #define `NOR_CMD_LUT_SEQ_IDX_RESTORE_NOCMD` 14U  
*Restore 0-4-4/0-8-8 mode sequence id in lookupTable stored in config block.*
- #define `NOR_CMD_LUT_SEQ_IDX_EXIT_NOCMD` 15U  
*Exit 0-4-4/0-8-8 mode sequence id in lookupTable stored in config block.*

## Enumerations

- enum {  
`kSerialFlash_ISSI_ManufacturerID = 0x9DU,`  
`kSerialFlash_Adesto_ManufacturerID = 0x1F,`  
`kSerialFlash_Winbond_ManufacturerID = 0xEFU,`  
`kSerialFlash_Cypress_ManufacturerID = 0x01U }`  
*Manufacturer ID.*
- enum `_flexspi_nor_status` {  
`kStatus_ROM_FLEXSPI_SequenceExecutionTimeout,`  
`kStatus_ROM_FLEXSPI_InvalidSequence = MAKE_STATUS(kROM_StatusGroup_FLEXSPI,`  
`1),`  
`kStatus_ROM_FLEXSPI_DeviceTimeout = MAKE_STATUS(kROM_StatusGroup_FLEXSPI, 2),`  
`kStatus_FLEXSPINOR_DTRRead_DummyProbeFailed,`  
`kStatus_ROM_FLEXSPINOR_SFDP_NotFound,`  
`kStatus_ROM_FLEXSPINOR_Flash_NotFound }`  
*ROM FLEXSPI NOR flash status.*
- enum `flexspi_operation_t` {  
`kFLEXSPIOperation_Command,`  
`kFLEXSPIOperation_Config,`  
`kFLEXSPIOperation_Write,`  
`kFLEXSPIOperation_Read }`

## Common ROMAPI features info defines

- #define `FSL_ROM_HAS_FLEXSPINOR_API` (1)
- #define `FSL_ROM_HAS_RUNBOOTLOADER_API` (0)
- #define `FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_GET_CONFIG` (0)
- #define `FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_FLASH_INIT` (1)



- #define **FSL\_ROM\_FLEXSPINOR\_API\_HAS\_FEATURE\_ERASE** (1)
- #define **FSL\_ROM\_FLEXSPINOR\_API\_HAS\_FEATURE\_ERASE\_SECTOR** (1)
- #define **FSL\_ROM\_FLEXSPINOR\_API\_HAS\_FEATURE\_ERASE\_BLOCK** (1)
- #define **FSL\_ROM\_FLEXSPINOR\_API\_HAS\_FEATURE\_ERASE\_ALL** (0)
- #define **FSL\_ROM\_FLEXSPINOR\_API\_HAS\_FEATURE\_PAGE\_PROGRAM** (1)
- #define **FSL\_ROM\_FLEXSPINOR\_API\_HAS\_FEATURE\_UPDATE\_LUT** (1)
- #define **FSL\_ROM\_FLEXSPINOR\_API\_HAS\_FEATURE\_CMD\_XFER** (1)

## Support for init FLEXSPI NOR configuration

- enum  
*Flash Pad Definitions.*
- enum {  
    **kFLEXSPIClk\_SDR**,  
    **kFLEXSPIClk\_DDR** }  
*FLEXSPI clock configuration type.*
- enum **flexspi\_read\_sample\_clk\_t**  
*FLEXSPI Read Sample Clock Source definition.*
- enum { **kFLEXSPIDeviceType\_SerialNOR** = 1U }  
*Flash Type Definition.*
- enum {  
    **kDeviceConfigCmdType\_Generic**,  
    **kDeviceConfigCmdType\_QuadEnable**,  
    **kDeviceConfigCmdType\_Spi2Xpi**,  
    **kDeviceConfigCmdType\_Xpi2Spi**,  
    **kDeviceConfigCmdType\_Spi2NoCmd**,  
    **kDeviceConfigCmdType\_Reset** }  
*Flash Configuration Command Type.*
- enum **flexspi\_serial\_clk\_freq\_t**  
*Defintions for FLEXSPI Serial Clock Frequency.*
- enum {  
    **kFLEXSPIMiscOffset\_DiffClkEnable** = 0U,  
    **kFLEXSPIMiscOffset\_Ck2Enable** = 1U,  
    **kFLEXSPIMiscOffset\_ParallelEnable** = 2U,  
    **kFLEXSPIMiscOffset\_WordAddressableEnable** = 3U,  
    **kFLEXSPIMiscOffset\_SafeConfigFreqEnable** = 4U,  
    **kFLEXSPIMiscOffset\_PadSettingOverrideEnable** = 5U,  
    **kFLEXSPIMiscOffset\_DdrModeEnable** = 6U,  
    **kFLEXSPIMiscOffset\_UseValidTimeForAllFreq** = 7U }  
*Misc feature bit definitions.*

## 5.2 Data Structure Documentation

### 5.2.1 struct flexspi\_lut\_seq\_t

#### Data Fields

- uint8\_t **seqNum**

- `uint8_t seqId`  
Sequence Number, valid number: 1-16.
- `uint8_t seqIdx`  
Sequence Index, valid number: 0-15.

## 5.2.2 struct flexspi\_mem\_config\_t

### Data Fields

- `uint32_t tag`  
*[0x000-0x003] Tag, fixed value 0x42464346UL*
- `uint32_t version`  
*[0x004-0x007] Version, [31:24] - 'V', [23:16] - Major, [15:8] - Minor, [7:0] - bugfix*
- `uint32_t reserved0`  
*[0x008-0x00b] Reserved for future use*
- `uint8_t readSampleClkSrc`  
*[0x00c-0x00c] Read Sample Clock Source, valid value: 0/1/3*
- `uint8_t csHoldTime`  
*[0x00d-0x00d] Data hold time, default value: 3*
- `uint8_t csSetupTime`  
*[0x00e-0x00e] Date setup time, default value: 3*
- `uint8_t columnAddressWidth`  
*[0x00f-0x00f] Column Address with, for HyperBus protocol, it is fixed to 3, For Serial NAND, need to refer to datasheet*
- `uint8_t deviceModeCfgEnable`  
*[0x010-0x010] Device Mode Configure enable flag, 1 - Enable, 0 - Disable*
- `uint8_t deviceModeType`  
*[0x011-0x011] Specify the configuration command type: Quad Enable, DPI/QPI/OPI switch, Generic configuration, etc.*
- `uint16_t waitTimeCfgCommands`  
*[0x012-0x013] Wait time for all configuration commands, unit: 100us, Used for DPI/QPI/OPI switch or reset command*
- `flexspi_lut_seq_t deviceModeSeq`  
*[0x014-0x017] Device mode sequence info, [7:0] - LUT sequence id, [15:8] - LUt sequence number, [31:16] Reserved*
- `uint32_t deviceModeArg`  
*[0x018-0x01b] Argument/Parameter for device configuration*
- `uint8_t configCmdEnable`  
*[0x01c-0x01c] Configure command Enable Flag, 1 - Enable, 0 - Disable*
- `uint8_t configModeType` [3]  
*[0x01d-0x01f] Configure Mode Type, similar as deviceModeTpe*
- `flexspi_lut_seq_t configCmdSeqs` [3]  
*[0x020-0x02b] Sequence info for Device Configuration command, similar as deviceModeSeq*
- `uint32_t reserved1`  
*[0x02c-0x02f] Reserved for future use*
- `uint32_t configCmdArgs` [3]  
*[0x030-0x03b] Arguments/Parameters for device Configuration commands*
- `uint32_t reserved2`  
*[0x03c-0x03f] Reserved for future use*
- `uint32_t controllerMiscOption`

- *[0x040-0x043] Controller Misc Options, see Misc feature bit definitions for more details*
- uint8\_t [deviceType](#)  
*[0x044-0x044] Device Type: See Flash Type Definition for more details*
- uint8\_t [sflashPadType](#)  
*[0x045-0x045] Serial Flash Pad Type: 1 - Single, 2 - Dual, 4 - Quad, 8 - Octal*
- uint8\_t [serialClkFreq](#)  
*[0x046-0x046] Serial Flash Frequency, device specific definitions, See System Boot Chapter for more details*
- uint8\_t [lutCustomSeqEnable](#)  
*[0x047-0x047] LUT customization Enable, it is required if the program/erase cannot be done using 1 LUT sequence, currently, only applicable to HyperFLASH*
- uint32\_t [reserved3](#) [2]  
*[0x048-0x04f] Reserved for future use*
- uint32\_t [sflashA1Size](#)  
*[0x050-0x053] Size of Flash connected to A1*
- uint32\_t [sflashA2Size](#)  
*[0x054-0x057] Size of Flash connected to A2*
- uint32\_t [sflashB1Size](#)  
*[0x058-0x05b] Size of Flash connected to B1*
- uint32\_t [sflashB2Size](#)  
*[0x05c-0x05f] Size of Flash connected to B2*
- uint32\_t [csPadSettingOverride](#)  
*[0x060-0x063] CS pad setting override value*
- uint32\_t [sclkPadSettingOverride](#)  
*[0x064-0x067] SCK pad setting override value*
- uint32\_t [dataPadSettingOverride](#)  
*[0x068-0x06b] data pad setting override value*
- uint32\_t [dqsPadSettingOverride](#)  
*[0x06c-0x06f] DQS pad setting override value*
- uint32\_t [timeoutInMs](#)  
*[0x070-0x073] Timeout threshold for read status command*
- uint32\_t [commandInterval](#)  
*[0x074-0x077] CS deselect interval between two commands*
- flexspi\_dll\_time\_t [dataValidTime](#) [2]  
*[0x078-0x07b] CLK edge to data valid time for PORT A and PORT B*
- uint16\_t [busyOffset](#)  
*[0x07c-0x07d] Busy offset, valid value: 0-31*
- uint16\_t [busyBitPolarity](#)  
*[0x07e-0x07f] Busy flag polarity, 0 - busy flag is 1 when flash device is busy, 1 - busy flag is 0 when flash device is busy*
- uint32\_t [lookupTable](#) [64]  
*[0x080-0x17f] Lookup table holds Flash command sequences*
- flexspi\_lut\_seq\_t [lutCustomSeq](#) [12]  
*[0x180-0x1af] Customizable LUT Sequences*
- uint32\_t [reserved4](#) [4]  
*[0x1b0-0x1bf] Reserved for future use*

## Field Documentation

(1) `uint8_t flexspi_mem_config_t::deviceModeType`

### 5.2.3 struct flexspi\_nor\_config\_t

#### Data Fields

- `flexspi_mem_config_t memConfig`  
*Common memory configuration info via FLEXSPI.*
- `uint32_t pageSize`  
*Page size of Serial NOR.*
- `uint32_t sectorSize`  
*Sector size of Serial NOR.*
- `uint8_t ipcmdSerialClkFreq`  
*Clock frequency for IP command.*
- `uint8_t isUniformBlockSize`  
*Sector/Block size is the same.*
- `uint8_t isDataOrderSwapped`  
*Data order (D0, D1, D2, D3) is swapped (D1,D0, D3, D2)*
- `uint8_t reserved0 [1]`  
*Reserved for future use.*
- `uint8_t serialNorType`  
*Serial NOR Flash type: 0/1/2/3.*
- `uint8_t needExitNoCmdMode`  
*Need to exit NoCmd mode before other IP command.*
- `uint8_t halfClkForNonReadCmd`  
*Half the Serial Clock for non-read command: true/false.*
- `uint8_t needRestoreNoCmdMode`  
*Need to Restore NoCmd mode after IP commmand execution.*
- `uint32_t blockSize`  
*Block size.*
- `uint32_t reserve2 [11]`  
*Reserved for future use.*

### 5.2.4 struct flexspi\_xfer\_t

#### Data Fields

- `flexspi_operation_t operation`  
*FLEXSPI operation.*
- `uint32_t baseAddress`  
*FLEXSPI operation base address.*
- `uint32_t seqId`  
*Sequence Id.*
- `uint32_t seqNum`  
*Sequence Number.*
- `bool isParallelModeEnable`

- *Is a parallel transfer.*
- `uint32_t * txBuffer`  
*Tx buffer.*
- `uint32_t txSize`  
*Tx size in bytes.*
- `uint32_t * rxBuffer`  
*Rx buffer.*
- `uint32_t rxSize`  
*Rx size in bytes.*

## 5.3 Macro Definition Documentation

5.3.1 `#define FSL_ROM_ROMAPI_VERSION (MAKE_VERSION(1U, 1U, 0U))`

5.3.2 `#define FSL_ROM_FLEXSPINOR_DRIVER_VERSION (MAKE_VERSION(1U, 4U, 0U))`

5.3.3 `#define kROM_StatusGroup_FLEXSPI 60U`

5.3.4 `#define kROM_StatusGroup_FLEXSPINOR 200U`

5.3.5 `#define FLEXSPI_CFG_BLK_TAG (0x42464346UL)`

ascii "FCFB" Big Endian

5.3.6 `#define NOR_CMD_LUT_SEQ_IDX_READ 0U`

READ LUT sequence id in lookupTable stored in config block

## 5.4 Enumeration Type Documentation

### 5.4.1 anonymous enum

Enumerator

- `kFLEXSPIClk_SDR` Clock configure for SDR mode.
- `kFLEXSPIClk_DDR` Clock configurat for DDR mode.

### 5.4.2 anonymous enum

Enumerator

- `kFLEXSPIDeviceType_SerialNOR` Flash device is Serial NOR.

### 5.4.3 anonymous enum

Enumerator

- kDeviceConfigCmdType\_Generic* Generic command, for example: configure dummy cycles, drive strength, etc.
- kDeviceConfigCmdType\_QuadEnable* Quad Enable command.
- kDeviceConfigCmdType\_Spi2Xpi* Switch from SPI to DPI/QPI/OPI mode.
- kDeviceConfigCmdType\_Xpi2Spi* Switch from DPI/QPI/OPI to SPI mode.
- kDeviceConfigCmdType\_Spi2NoCmd* Switch to 0-4-4/0-8-8 mode.
- kDeviceConfigCmdType\_Reset* Reset device command.

### 5.4.4 anonymous enum

Enumerator

- kFLEXSPIMiscOffset\_DiffClkEnable* Bit for Differential clock enable.
- kFLEXSPIMiscOffset\_Ck2Enable* Bit for CK2 enable.
- kFLEXSPIMiscOffset\_ParallelEnable* Bit for Parallel mode enable.
- kFLEXSPIMiscOffset\_WordAddressableEnable* Bit for Word Addressable enable.
- kFLEXSPIMiscOffset\_SafeConfigFreqEnable* Bit for Safe Configuration Frequency enable.
- kFLEXSPIMiscOffset\_PadSettingOverrideEnable* Bit for Pad setting override enable.
- kFLEXSPIMiscOffset\_DdrModeEnable* Bit for DDR clock configuration indication.
- kFLEXSPIMiscOffset\_UseValidTimeForAllFreq* Bit for DLLCR settings under all modes.

### 5.4.5 anonymous enum

Enumerator

- kSerialFlash\_ISSI\_ManufacturerID* Manufacturer ID of the ISSI serial flash.
- kSerialFlash\_Adesto\_ManufacturerID* Manufacturer ID of the Adesto Technologies serial flash.
- kSerialFlash\_Winbond\_ManufacturerID* Manufacturer ID of the Winbond serial flash.
- kSerialFlash\_Cypress\_ManufacturerID* Manufacturer ID for Cypress.

### 5.4.6 enum \_flexspi\_nor\_status

Enumerator

- kStatus\_ROM\_FLEXSPI\_SequenceExecutionTimeout* Status for Sequence Execution timeout.
- kStatus\_ROM\_FLEXSPI\_InvalidSequence* Status for Invalid Sequence.
- kStatus\_ROM\_FLEXSPI\_DeviceTimeout* Status for Device timeout.
- kStatus\_FLEXSPINOR\_DTRRead\_DummyProbeFailed* Status for DDR Read dummy probe failure.

*kStatus\_ROM\_FLEXSPINOR\_SFDP\_NotFound* Status for SFDP read failure.

*kStatus\_ROM\_FLEXSPINOR\_Flash\_NotFound* Status for Flash detection failure.

### 5.4.7 enum flexspi\_operation\_t

Enumerator

*kFLEXSPIOperation\_Command* FLEXSPI operation: Only command, both TX and RX buffer are ignored.

*kFLEXSPIOperation\_Config* FLEXSPI operation: Configure device mode, the TX FIFO size is fixed in LUT.

*kFLEXSPIOperation\_Write* FLEXSPI operation: Write, only TX buffer is effective.

*kFLEXSPIOperation\_Read* FLEXSPI operation: Read, only Rx Buffer is effective.

# Chapter 6

## IOMUXC: IOMUX Controller

### 6.1 Overview

IOMUXC driver provides APIs for pin configuration. It also supports the miscellaneous functions integrated in IOMUXC.

#### Files

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

#### Driver version

- #define [FSL\\_IOMUXC\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 2))  
*IOMUXC driver version 2.0.2.*

#### Pin function ID

The pin function ID is a tuple of <muxRegister muxMode inputRegister inputDaisy configRegister>

- #define [IOMUXC\\_GPIO\\_EMC\\_00\\_SEMC\\_DATA00](#) 0x401F8014U, 0x0U, 0, 0, 0x401F8188U
- #define [IOMUXC\\_GPIO\\_EMC\\_00\\_QTIMER2\\_TIMER0](#) 0x401F8014U, 0x1U, 0x401F8420U, 0x0U, 0x401F8188U
- #define [IOMUXC\\_GPIO\\_EMC\\_00\\_LPUART4\\_CTS\\_B](#) 0x401F8014U, 0x2U, 0x401F83E0U, 0x0U, 0x401F8188U
- #define [IOMUXC\\_GPIO\\_EMC\\_00\\_SPDIF\\_SR\\_CLK](#) 0x401F8014U, 0x3U, 0, 0, 0x401F8188U
- #define [IOMUXC\\_GPIO\\_EMC\\_00\\_LPSPI2\\_SCK](#) 0x401F8014U, 0x4U, 0, 0, 0x401F8188U
- #define [IOMUXC\\_GPIO\\_EMC\\_00\\_GPIO2\\_IO00](#) 0x401F8014U, 0x5U, 0, 0, 0x401F8188U
- #define [IOMUXC\\_GPIO\\_EMC\\_00\\_FLEXCAN1\\_TX](#) 0x401F8014U, 0x6U, 0, 0, 0x401F8188U
- #define [IOMUXC\\_GPIO\\_EMC\\_00\\_PIT\\_TRIGGER02](#) 0x401F8014U, 0x7U, 0, 0, 0x401F8188U
- #define [IOMUXC\\_GPIO\\_EMC\\_01\\_SEMC\\_DATA01](#) 0x401F8018U, 0x0U, 0, 0, 0x401F818CU
- #define [IOMUXC\\_GPIO\\_EMC\\_01\\_QTIMER2\\_TIMER1](#) 0x401F8018U, 0x1U, 0x401F8424U, 0x0U, 0x401F818CU
- #define [IOMUXC\\_GPIO\\_EMC\\_01\\_LPUART4\\_RTS\\_B](#) 0x401F8018U, 0x2U, 0, 0, 0x401F818CU
- #define [IOMUXC\\_GPIO\\_EMC\\_01\\_SPDIF\\_OUT](#) 0x401F8018U, 0x3U, 0, 0, 0x401F818CU
- #define [IOMUXC\\_GPIO\\_EMC\\_01\\_LPSPI2\\_PCS0](#) 0x401F8018U, 0x4U, 0x401F83ACU, 0x0U, 0x401F818CU
- #define [IOMUXC\\_GPIO\\_EMC\\_01\\_GPIO2\\_IO01](#) 0x401F8018U, 0x5U, 0, 0, 0x401F818CU
- #define [IOMUXC\\_GPIO\\_EMC\\_01\\_FLEXCAN1\\_RX](#) 0x401F8018U, 0x6U, 0x401F8320U, 0x0U, 0x401F818CU
- #define [IOMUXC\\_GPIO\\_EMC\\_01\\_PIT\\_TRIGGER03](#) 0x401F8018U, 0x7U, 0, 0, 0x401F818CU
- #define [IOMUXC\\_GPIO\\_EMC\\_02\\_SEMC\\_DATA02](#) 0x401F801CU, 0x0U, 0, 0, 0x401F8190U
- #define [IOMUXC\\_GPIO\\_EMC\\_02\\_QTIMER2\\_TIMER2](#) 0x401F801CU, 0x1U, 0x401F8428U, 0x0U, 0x401F8190U



- #define **IOMUXC\_GPIO\_EMC\_02\_LPUART4\_TX** 0x401F801CU, 0x2U, 0x401F83E8U, 0x0U, 0x401F8190U
- #define **IOMUXC\_GPIO\_EMC\_02\_SPDIF\_LOCK** 0x401F801CU, 0x3U, 0, 0, 0x401F8190U
- #define **IOMUXC\_GPIO\_EMC\_02\_LPSPI2\_SDO** 0x401F801CU, 0x4U, 0x401F83B8U, 0x0U, 0x401F8190U
- #define **IOMUXC\_GPIO\_EMC\_02\_GPIO2\_IO02** 0x401F801CU, 0x5U, 0, 0, 0x401F8190U
- #define **IOMUXC\_GPIO\_EMC\_02\_LPI2C1\_SCL** 0x401F801CU, 0x6U, 0x401F837CU, 0x0U, 0x401F8190U
- #define **IOMUXC\_GPIO\_EMC\_03\_SEMC\_DATA03** 0x401F8020U, 0x0U, 0, 0, 0x401F8194U
- #define **IOMUXC\_GPIO\_EMC\_03\_QTIMER2\_TIMER3** 0x401F8020U, 0x1U, 0x401F842CU, 0x0U, 0x401F8194U
- #define **IOMUXC\_GPIO\_EMC\_03\_LPUART4\_RX** 0x401F8020U, 0x2U, 0x401F83E4U, 0x0U, 0x401F8194U
- #define **IOMUXC\_GPIO\_EMC\_03\_SPDIF\_EXT\_CLK** 0x401F8020U, 0x3U, 0, 0, 0x401F8194U
- #define **IOMUXC\_GPIO\_EMC\_03\_LPSPI2\_SDI** 0x401F8020U, 0x4U, 0x401F83B4U, 0x0U, 0x401F8194U
- #define **IOMUXC\_GPIO\_EMC\_03\_GPIO2\_IO03** 0x401F8020U, 0x5U, 0, 0, 0x401F8194U
- #define **IOMUXC\_GPIO\_EMC\_03\_LPI2C1\_SDA** 0x401F8020U, 0x6U, 0x401F8380U, 0x0U, 0x401F8194U
- #define **IOMUXC\_GPIO\_EMC\_04\_SEMC\_DATA04** 0x401F8024U, 0x0U, 0, 0, 0x401F8198U
- #define **IOMUXC\_GPIO\_EMC\_04\_XBAR1\_INOUT04** 0x401F8024U, 0x1U, 0, 0, 0x401F8198U
- #define **IOMUXC\_GPIO\_EMC\_04\_SPDIF\_OUT** 0x401F8024U, 0x2U, 0, 0, 0x401F8198U
- #define **IOMUXC\_GPIO\_EMC\_04\_SAI2\_TX\_BCLK** 0x401F8024U, 0x3U, 0x401F8464U, 0x1U, 0x401F8198U
- #define **IOMUXC\_GPIO\_EMC\_04\_FLEXIO1\_FLEXIO16** 0x401F8024U, 0x4U, 0, 0, 0x401F8198U
- #define **IOMUXC\_GPIO\_EMC\_04\_GPIO2\_IO04** 0x401F8024U, 0x5U, 0, 0, 0x401F8198U
- #define **IOMUXC\_GPIO\_EMC\_05\_SEMC\_DATA05** 0x401F8028U, 0x0U, 0, 0, 0x401F819CU
- #define **IOMUXC\_GPIO\_EMC\_05\_XBAR1\_INOUT05** 0x401F8028U, 0x1U, 0, 0, 0x401F819CU
- #define **IOMUXC\_GPIO\_EMC\_05\_SPDIF\_IN** 0x401F8028U, 0x2U, 0x401F8488U, 0x0U, 0x401F819CU
- #define **IOMUXC\_GPIO\_EMC\_05\_SAI2\_TX\_SYNC** 0x401F8028U, 0x3U, 0x401F8468U, 0x1U, 0x401F819CU
- #define **IOMUXC\_GPIO\_EMC\_05\_FLEXIO1\_FLEXIO17** 0x401F8028U, 0x4U, 0, 0, 0x401F819CU
- #define **IOMUXC\_GPIO\_EMC\_05\_GPIO2\_IO05** 0x401F8028U, 0x5U, 0, 0, 0x401F819CU
- #define **IOMUXC\_GPIO\_EMC\_06\_SEMC\_DATA06** 0x401F802CU, 0x0U, 0, 0, 0x401F81A0U
- #define **IOMUXC\_GPIO\_EMC\_06\_XBAR1\_INOUT06** 0x401F802CU, 0x1U, 0, 0, 0x401F81A0U
- #define **IOMUXC\_GPIO\_EMC\_06\_LPUART3\_TX** 0x401F802CU, 0x2U, 0x401F83DCU, 0x0U, 0x401F81A0U
- #define **IOMUXC\_GPIO\_EMC\_06\_SAI2\_TX\_DATA** 0x401F802CU, 0x3U, 0, 0, 0x401F81A0U
- #define **IOMUXC\_GPIO\_EMC\_06\_FLEXIO1\_FLEXIO18** 0x401F802CU, 0x4U, 0, 0, 0x401F81A0U
- #define **IOMUXC\_GPIO\_EMC\_06\_GPIO2\_IO06** 0x401F802CU, 0x5U, 0, 0, 0x401F81A0U
- #define **IOMUXC\_GPIO\_EMC\_07\_SEMC\_DATA07** 0x401F8030U, 0x0U, 0, 0, 0x401F81A4U
- #define **IOMUXC\_GPIO\_EMC\_07\_XBAR1\_INOUT07** 0x401F8030U, 0x1U, 0, 0, 0x401F81A4U

- A4U
- #define **IOMUXC\_GPIO\_EMC\_07\_LPUART3\_RX** 0x401F8030U, 0x2U, 0x401F83D8U, 0x0U, 0x401F81A4U
- #define **IOMUXC\_GPIO\_EMC\_07\_SAI2\_RX\_SYNC** 0x401F8030U, 0x3U, 0x401F8460U, 0x1U, 0x401F81A4U
- #define **IOMUXC\_GPIO\_EMC\_07\_FLEXIO1\_FLEXIO19** 0x401F8030U, 0x4U, 0, 0, 0x401F81A4U
- #define **IOMUXC\_GPIO\_EMC\_07\_GPIO2\_IO07** 0x401F8030U, 0x5U, 0, 0, 0x401F81A4U
- #define **IOMUXC\_GPIO\_EMC\_08\_SEMC\_DM00** 0x401F8034U, 0x0U, 0, 0, 0x401F81A8U
- #define **IOMUXC\_GPIO\_EMC\_08\_XBAR1\_INOUT08** 0x401F8034U, 0x1U, 0, 0, 0x401F81A8U
- #define **IOMUXC\_GPIO\_EMC\_08\_FLEXCAN2\_TX** 0x401F8034U, 0x2U, 0, 0, 0x401F81A8U
- #define **IOMUXC\_GPIO\_EMC\_08\_SAI2\_RX\_DATA** 0x401F8034U, 0x3U, 0x401F845CU, 0x1U, 0x401F81A8U
- #define **IOMUXC\_GPIO\_EMC\_08\_FLEXIO1\_FLEXIO20** 0x401F8034U, 0x4U, 0, 0, 0x401F81A8U
- #define **IOMUXC\_GPIO\_EMC\_08\_GPIO2\_IO08** 0x401F8034U, 0x5U, 0, 0, 0x401F81A8U
- #define **IOMUXC\_GPIO\_EMC\_09\_SEMC\_WE** 0x401F8038U, 0x0U, 0, 0, 0x401F81ACU
- #define **IOMUXC\_GPIO\_EMC\_09\_XBAR1\_INOUT09** 0x401F8038U, 0x1U, 0, 0, 0x401F81ACU
- #define **IOMUXC\_GPIO\_EMC\_09\_FLEXCAN2\_RX** 0x401F8038U, 0x2U, 0x401F8324U, 0x1U, 0x401F81ACU
- #define **IOMUXC\_GPIO\_EMC\_09\_SAI2\_RX\_BCLK** 0x401F8038U, 0x3U, 0x401F8458U, 0x1U, 0x401F81ACU
- #define **IOMUXC\_GPIO\_EMC\_09\_FLEXIO1\_FLEXIO21** 0x401F8038U, 0x4U, 0, 0, 0x401F81ACU
- #define **IOMUXC\_GPIO\_EMC\_09\_GPIO2\_IO09** 0x401F8038U, 0x5U, 0, 0, 0x401F81ACU
- #define **IOMUXC\_GPIO\_EMC\_10\_SEMC\_CAS** 0x401F803CU, 0x0U, 0, 0, 0x401F81B0U
- #define **IOMUXC\_GPIO\_EMC\_10\_XBAR1\_INOUT10** 0x401F803CU, 0x1U, 0x401F84B0U, 0x0U, 0x401F81B0U
- #define **IOMUXC\_GPIO\_EMC\_10\_LPI2C4\_SDA** 0x401F803CU, 0x2U, 0x401F8398U, 0x0U, 0x401F81B0U
- #define **IOMUXC\_GPIO\_EMC\_10\_SAI1\_TX\_SYNC** 0x401F803CU, 0x3U, 0x401F8450U, 0x0U, 0x401F81B0U
- #define **IOMUXC\_GPIO\_EMC\_10\_LPSPI2\_SCK** 0x401F803CU, 0x4U, 0, 0, 0x401F81B0U
- #define **IOMUXC\_GPIO\_EMC\_10\_GPIO2\_IO10** 0x401F803CU, 0x5U, 0, 0, 0x401F81B0U
- #define **IOMUXC\_GPIO\_EMC\_10\_FLEXPWM2\_PWMX00** 0x401F803CU, 0x6U, 0, 0, 0x401F81B0U
- #define **IOMUXC\_GPIO\_EMC\_11\_SEMC\_RAS** 0x401F8040U, 0x0U, 0, 0, 0x401F81B4U
- #define **IOMUXC\_GPIO\_EMC\_11\_XBAR1\_INOUT11** 0x401F8040U, 0x1U, 0, 0, 0x401F81B4U
- #define **IOMUXC\_GPIO\_EMC\_11\_LPI2C4\_SCL** 0x401F8040U, 0x2U, 0x401F8394U, 0x0U, 0x401F81B4U
- #define **IOMUXC\_GPIO\_EMC\_11\_SAI1\_TX\_BCLK** 0x401F8040U, 0x3U, 0x401F844CU, 0x0U, 0x401F81B4U
- #define **IOMUXC\_GPIO\_EMC\_11\_LPSPI2\_PCS0** 0x401F8040U, 0x4U, 0x401F83ACU, 0x1U, 0x401F81B4U
- #define **IOMUXC\_GPIO\_EMC\_11\_GPIO2\_IO11** 0x401F8040U, 0x5U, 0, 0, 0x401F81B4U
- #define **IOMUXC\_GPIO\_EMC\_11\_FLEXPWM2\_PWMX01** 0x401F8040U, 0x6U, 0, 0, 0x401F81B4U
- #define **IOMUXC\_GPIO\_EMC\_12\_SEMC\_CS0** 0x401F8044U, 0x0U, 0, 0, 0x401F81B8U

- #define **IOMUXC\_GPIO\_EMCC\_12\_XBAR1\_INOUT12** 0x401F8044U, 0x1U, 0x401F84B4U, 0x0U, 0x401F81B8U
- #define **IOMUXC\_GPIO\_EMCC\_12\_LPUART6\_TX** 0x401F8044U, 0x2U, 0x401F83F8U, 0x0U, 0x401F81B8U
- #define **IOMUXC\_GPIO\_EMCC\_12\_SAI1\_TX\_DATA00** 0x401F8044U, 0x3U, 0, 0, 0x401F81B8U
- #define **IOMUXC\_GPIO\_EMCC\_12\_LPSPI2\_SDO** 0x401F8044U, 0x4U, 0x401F83B8U, 0x1U, 0x401F81B8U
- #define **IOMUXC\_GPIO\_EMCC\_12\_GPIO2\_IO12** 0x401F8044U, 0x5U, 0, 0, 0x401F81B8U
- #define **IOMUXC\_GPIO\_EMCC\_12\_FLEXPWM2\_PWMX02** 0x401F8044U, 0x6U, 0, 0, 0x401F81B8U
- #define **IOMUXC\_GPIO\_EMCC\_13\_SEMCC\_BA0** 0x401F8048U, 0x0U, 0, 0, 0x401F81BCU
- #define **IOMUXC\_GPIO\_EMCC\_13\_XBAR1\_INOUT13** 0x401F8048U, 0x1U, 0x401F84B8U, 0x0U, 0x401F81BCU
- #define **IOMUXC\_GPIO\_EMCC\_13\_LPUART6\_RX** 0x401F8048U, 0x2U, 0x401F83F4U, 0x0U, 0x401F81BCU
- #define **IOMUXC\_GPIO\_EMCC\_13\_SAI1\_RX\_DATA00** 0x401F8048U, 0x3U, 0x401F8438U, 0x0U, 0x401F81BCU
- #define **IOMUXC\_GPIO\_EMCC\_13\_LPSPI2\_SDI** 0x401F8048U, 0x4U, 0x401F83B4U, 0x1U, 0x401F81BCU
- #define **IOMUXC\_GPIO\_EMCC\_13\_GPIO2\_IO13** 0x401F8048U, 0x5U, 0, 0, 0x401F81BCU
- #define **IOMUXC\_GPIO\_EMCC\_13\_FLEXPWM2\_PWMX03** 0x401F8048U, 0x6U, 0, 0, 0x401F81BCU
- #define **IOMUXC\_GPIO\_EMCC\_13\_CCM\_PMIC\_RDY** 0x401F8048U, 0x7U, 0x401F8300U, 0x0U, 0x401F81BCU
- #define **IOMUXC\_GPIO\_EMCC\_14\_SEMCC\_BA1** 0x401F804CU, 0x0U, 0, 0, 0x401F81C0U
- #define **IOMUXC\_GPIO\_EMCC\_14\_XBAR1\_INOUT14** 0x401F804CU, 0x1U, 0x401F84A0U, 0x1U, 0x401F81C0U
- #define **IOMUXC\_GPIO\_EMCC\_14\_LPUART6\_CTS\_B** 0x401F804CU, 0x2U, 0, 0, 0x401F81C0U
- #define **IOMUXC\_GPIO\_EMCC\_14\_SAI1\_RX\_BCLK** 0x401F804CU, 0x3U, 0x401F8434U, 0x1U, 0x401F81C0U
- #define **IOMUXC\_GPIO\_EMCC\_14\_LPSPI2\_PCS1** 0x401F804CU, 0x4U, 0, 0, 0x401F81C0U
- #define **IOMUXC\_GPIO\_EMCC\_14\_GPIO2\_IO14** 0x401F804CU, 0x5U, 0, 0, 0x401F81C0U
- #define **IOMUXC\_GPIO\_EMCC\_14\_FLEXPWM2\_TX** 0x401F804CU, 0x6U, 0, 0, 0x401F81C0U
- #define **IOMUXC\_GPIO\_EMCC\_15\_SEMCC\_ADDR10** 0x401F8050U, 0x0U, 0, 0, 0x401F81C4U
- #define **IOMUXC\_GPIO\_EMCC\_15\_XBAR1\_INOUT15** 0x401F8050U, 0x1U, 0x401F84A4U, 0x1U, 0x401F81C4U
- #define **IOMUXC\_GPIO\_EMCC\_15\_LPUART6\_RTS\_B** 0x401F8050U, 0x2U, 0, 0, 0x401F81C4U
- #define **IOMUXC\_GPIO\_EMCC\_15\_SAI1\_RX\_SYNC** 0x401F8050U, 0x3U, 0x401F8448U, 0x1U, 0x401F81C4U
- #define **IOMUXC\_GPIO\_EMCC\_15\_WDOG1\_B** 0x401F8050U, 0x4U, 0, 0, 0x401F81C4U
- #define **IOMUXC\_GPIO\_EMCC\_15\_GPIO2\_IO15** 0x401F8050U, 0x5U, 0, 0, 0x401F81C4U
- #define **IOMUXC\_GPIO\_EMCC\_15\_FLEXPWM2\_RX** 0x401F8050U, 0x6U, 0x401F8320U, 0x3U, 0x401F81C4U
- #define **IOMUXC\_GPIO\_EMCC\_16\_SEMCC\_ADDR00** 0x401F8054U, 0x0U, 0, 0, 0x401F81C8U
- #define **IOMUXC\_GPIO\_EMCC\_16\_MQS\_RIGHT** 0x401F8054U, 0x2U, 0, 0, 0x401F81C8U
- #define **IOMUXC\_GPIO\_EMCC\_16\_SAI2\_MCLK** 0x401F8054U, 0x3U, 0x401F8454U, 0x1U, 0x401F81C8U
- #define **IOMUXC\_GPIO\_EMCC\_16\_GPIO2\_IO16** 0x401F8054U, 0x5U, 0, 0, 0x401F81C8U

- #define **IOMUXC\_GPIO\_EMC\_16\_SRC\_BOOT\_MODE00** 0x401F8054U, 0x6U, 0, 0, 0x401F81C8U
- #define **IOMUXC\_GPIO\_EMC\_17\_SEMC\_ADDR01** 0x401F8058U, 0x0U, 0, 0, 0x401F81CCU
- #define **IOMUXC\_GPIO\_EMC\_17\_MQS\_LEFT** 0x401F8058U, 0x2U, 0, 0, 0x401F81CCU
- #define **IOMUXC\_GPIO\_EMC\_17\_SAI3\_MCLK** 0x401F8058U, 0x3U, 0x401F846CU, 0x1U, 0x401F81CCU
- #define **IOMUXC\_GPIO\_EMC\_17\_GPIO2\_IO17** 0x401F8058U, 0x5U, 0, 0, 0x401F81CCU
- #define **IOMUXC\_GPIO\_EMC\_17\_SRC\_BOOT\_MODE01** 0x401F8058U, 0x6U, 0, 0, 0x401F81CCU
- #define **IOMUXC\_GPIO\_EMC\_18\_SEMC\_ADDR02** 0x401F805CU, 0x0U, 0, 0, 0x401F81D0U
- #define **IOMUXC\_GPIO\_EMC\_18\_XBAR1\_INOUT16** 0x401F805CU, 0x1U, 0x401F84A8U, 0x1U, 0x401F81D0U
- #define **IOMUXC\_GPIO\_EMC\_18\_LPI2C2\_SDA** 0x401F805CU, 0x2U, 0x401F8388U, 0x1U, 0x401F81D0U
- #define **IOMUXC\_GPIO\_EMC\_18\_SAI1\_RX\_SYNC** 0x401F805CU, 0x3U, 0x401F8448U, 0x2U, 0x401F81D0U
- #define **IOMUXC\_GPIO\_EMC\_18\_FLEXIO1\_FLEXIO22** 0x401F805CU, 0x4U, 0, 0, 0x401F81D0U
- #define **IOMUXC\_GPIO\_EMC\_18\_GPIO2\_IO18** 0x401F805CU, 0x5U, 0, 0, 0x401F81D0U
- #define **IOMUXC\_GPIO\_EMC\_18\_SRC\_BT\_CFG00** 0x401F805CU, 0x6U, 0, 0, 0x401F81D0U
- #define **IOMUXC\_GPIO\_EMC\_19\_SEMC\_ADDR03** 0x401F8060U, 0x0U, 0, 0, 0x401F81D4U
- #define **IOMUXC\_GPIO\_EMC\_19\_XBAR1\_INOUT17** 0x401F8060U, 0x1U, 0x401F84ACU, 0x1U, 0x401F81D4U
- #define **IOMUXC\_GPIO\_EMC\_19\_LPI2C2\_SCL** 0x401F8060U, 0x2U, 0x401F8384U, 0x1U, 0x401F81D4U
- #define **IOMUXC\_GPIO\_EMC\_19\_SAI1\_RX\_BCLK** 0x401F8060U, 0x3U, 0x401F8434U, 0x2U, 0x401F81D4U
- #define **IOMUXC\_GPIO\_EMC\_19\_FLEXIO1\_FLEXIO23** 0x401F8060U, 0x4U, 0, 0, 0x401F81D4U
- #define **IOMUXC\_GPIO\_EMC\_19\_GPIO2\_IO19** 0x401F8060U, 0x5U, 0, 0, 0x401F81D4U
- #define **IOMUXC\_GPIO\_EMC\_19\_SRC\_BT\_CFG01** 0x401F8060U, 0x6U, 0, 0, 0x401F81D4U
- #define **IOMUXC\_GPIO\_EMC\_20\_SEMC\_ADDR04** 0x401F8064U, 0x0U, 0, 0, 0x401F81D8U
- #define **IOMUXC\_GPIO\_EMC\_20\_FLEXPWM1\_PWMA03** 0x401F8064U, 0x1U, 0x401F8334U, 0x1U, 0x401F81D8U
- #define **IOMUXC\_GPIO\_EMC\_20\_LPUART2\_CTS\_B** 0x401F8064U, 0x2U, 0x401F83CCU, 0x1U, 0x401F81D8U
- #define **IOMUXC\_GPIO\_EMC\_20\_SAI1\_MCLK** 0x401F8064U, 0x3U, 0x401F8430U, 0x3U, 0x401F81D8U
- #define **IOMUXC\_GPIO\_EMC\_20\_FLEXIO1\_FLEXIO24** 0x401F8064U, 0x4U, 0, 0, 0x401F81D8U
- #define **IOMUXC\_GPIO\_EMC\_20\_GPIO2\_IO20** 0x401F8064U, 0x5U, 0, 0, 0x401F81D8U
- #define **IOMUXC\_GPIO\_EMC\_20\_SRC\_BT\_CFG02** 0x401F8064U, 0x6U, 0, 0, 0x401F81D8U
- #define **IOMUXC\_GPIO\_EMC\_21\_SEMC\_ADDR05** 0x401F8068U, 0x0U, 0, 0, 0x401F81DCU
- #define **IOMUXC\_GPIO\_EMC\_21\_FLEXPWM1\_PWMB03** 0x401F8068U, 0x1U, 0x401F8344U, 0x1U, 0x401F81DCU
- #define **IOMUXC\_GPIO\_EMC\_21\_LPUART2\_RTS\_B** 0x401F8068U, 0x2U, 0, 0, 0x401F81DCU

- DCU
- #define **IOMUXC\_GPIO\_EMC\_21\_SAI1\_RX\_DATA00** 0x401F8068U, 0x3U, 0x401F8438U, 0x2U, 0x401F81DCU
  - #define **IOMUXC\_GPIO\_EMC\_21\_FLEXIO1\_FLEXIO25** 0x401F8068U, 0x4U, 0, 0, 0x401F81DCU
  - #define **IOMUXC\_GPIO\_EMC\_21\_GPIO2\_IO21** 0x401F8068U, 0x5U, 0, 0, 0x401F81DCU
  - #define **IOMUXC\_GPIO\_EMC\_21\_SRC\_BT\_CFG03** 0x401F8068U, 0x6U, 0, 0, 0x401F81DCU
  - #define **IOMUXC\_GPIO\_EMC\_22\_SEMC\_ADDR06** 0x401F806CU, 0x0U, 0, 0, 0x401F81E0U
  - #define **IOMUXC\_GPIO\_EMC\_22\_FLEXPWM1\_PWMA02** 0x401F806CU, 0x1U, 0x401F8330U, 0x1U, 0x401F81E0U
  - #define **IOMUXC\_GPIO\_EMC\_22\_LPUART2\_TX** 0x401F806CU, 0x2U, 0x401F83D4U, 0x1U, 0x401F81E0U
  - #define **IOMUXC\_GPIO\_EMC\_22\_SAI1\_TX\_DATA03** 0x401F806CU, 0x3U, 0x401F843CU, 0x1U, 0x401F81E0U
  - #define **IOMUXC\_GPIO\_EMC\_22\_FLEXIO1\_FLEXIO26** 0x401F806CU, 0x4U, 0, 0, 0x401F81E0U
  - #define **IOMUXC\_GPIO\_EMC\_22\_GPIO2\_IO22** 0x401F806CU, 0x5U, 0, 0, 0x401F81E0U
  - #define **IOMUXC\_GPIO\_EMC\_22\_SRC\_BT\_CFG04** 0x401F806CU, 0x6U, 0, 0, 0x401F81E0U
  - #define **IOMUXC\_GPIO\_EMC\_23\_SEMC\_ADDR07** 0x401F8070U, 0x0U, 0, 0, 0x401F81E4U
  - #define **IOMUXC\_GPIO\_EMC\_23\_FLEXPWM1\_PWMB02** 0x401F8070U, 0x1U, 0x401F8340U, 0x1U, 0x401F81E4U
  - #define **IOMUXC\_GPIO\_EMC\_23\_LPUART2\_RX** 0x401F8070U, 0x2U, 0x401F83D0U, 0x1U, 0x401F81E4U
  - #define **IOMUXC\_GPIO\_EMC\_23\_SAI1\_TX\_DATA02** 0x401F8070U, 0x3U, 0x401F8440U, 0x1U, 0x401F81E4U
  - #define **IOMUXC\_GPIO\_EMC\_23\_FLEXIO1\_FLEXIO27** 0x401F8070U, 0x4U, 0, 0, 0x401F81E4U
  - #define **IOMUXC\_GPIO\_EMC\_23\_GPIO2\_IO23** 0x401F8070U, 0x5U, 0, 0, 0x401F81E4U
  - #define **IOMUXC\_GPIO\_EMC\_23\_SRC\_BT\_CFG05** 0x401F8070U, 0x6U, 0, 0, 0x401F81E4U
  - #define **IOMUXC\_GPIO\_EMC\_24\_SEMC\_ADDR08** 0x401F8074U, 0x0U, 0, 0, 0x401F81E8U
  - #define **IOMUXC\_GPIO\_EMC\_24\_FLEXPWM1\_PWMA01** 0x401F8074U, 0x1U, 0x401F832CU, 0x1U, 0x401F81E8U
  - #define **IOMUXC\_GPIO\_EMC\_24\_LPUART8\_CTS\_B** 0x401F8074U, 0x2U, 0, 0, 0x401F81E8U
  - #define **IOMUXC\_GPIO\_EMC\_24\_SAI1\_TX\_DATA01** 0x401F8074U, 0x3U, 0x401F8444U, 0x1U, 0x401F81E8U
  - #define **IOMUXC\_GPIO\_EMC\_24\_FLEXIO1\_FLEXIO28** 0x401F8074U, 0x4U, 0, 0, 0x401F81E8U
  - #define **IOMUXC\_GPIO\_EMC\_24\_GPIO2\_IO24** 0x401F8074U, 0x5U, 0, 0, 0x401F81E8U
  - #define **IOMUXC\_GPIO\_EMC\_24\_SRC\_BT\_CFG06** 0x401F8074U, 0x6U, 0, 0, 0x401F81E8U
  - #define **IOMUXC\_GPIO\_EMC\_25\_SEMC\_ADDR09** 0x401F8078U, 0x0U, 0, 0, 0x401F81ECU
  - #define **IOMUXC\_GPIO\_EMC\_25\_FLEXPWM1\_PWMB01** 0x401F8078U, 0x1U, 0x401F833CU, 0x1U, 0x401F81ECU
  - #define **IOMUXC\_GPIO\_EMC\_25\_LPUART8\_RTS\_B** 0x401F8078U, 0x2U, 0, 0, 0x401F81ECU
  - #define **IOMUXC\_GPIO\_EMC\_25\_SAI1\_TX\_DATA00** 0x401F8078U, 0x3U, 0, 0, 0x401F81ECU
  - #define **IOMUXC\_GPIO\_EMC\_25\_FLEXIO1\_FLEXIO29** 0x401F8078U, 0x4U, 0, 0, 0x401F81ECU

- #define **IOMUXC\_GPIO\_EMC\_25\_GPIO2\_IO25** 0x401F8078U, 0x5U, 0, 0, 0x401F81ECU
- #define **IOMUXC\_GPIO\_EMC\_25\_SRC\_BT\_CFG07** 0x401F8078U, 0x6U, 0, 0, 0x401F81ECU
- #define **IOMUXC\_GPIO\_EMC\_26\_SEMC\_ADDR11** 0x401F807CU, 0x0U, 0, 0, 0x401F81F0U
- #define **IOMUXC\_GPIO\_EMC\_26\_FLEXPWM1\_PWMA00** 0x401F807CU, 0x1U, 0x401F8328U, 0x1U, 0x401F81F0U
- #define **IOMUXC\_GPIO\_EMC\_26\_LPUART8\_TX** 0x401F807CU, 0x2U, 0x401F8408U, 0x1U, 0x401F81F0U
- #define **IOMUXC\_GPIO\_EMC\_26\_SAI1\_TX\_BCLK** 0x401F807CU, 0x3U, 0x401F844CU, 0x2U, 0x401F81F0U
- #define **IOMUXC\_GPIO\_EMC\_26\_FLEXIO1\_FLEXIO30** 0x401F807CU, 0x4U, 0, 0, 0x401F81F0U
- #define **IOMUXC\_GPIO\_EMC\_26\_GPIO2\_IO26** 0x401F807CU, 0x5U, 0, 0, 0x401F81F0U
- #define **IOMUXC\_GPIO\_EMC\_26\_SRC\_BT\_CFG08** 0x401F807CU, 0x6U, 0, 0, 0x401F81F0U
- #define **IOMUXC\_GPIO\_EMC\_27\_SEMC\_ADDR12** 0x401F8080U, 0x0U, 0, 0, 0x401F81F4U
- #define **IOMUXC\_GPIO\_EMC\_27\_FLEXPWM1\_PWMB00** 0x401F8080U, 0x1U, 0x401F8338U, 0x1U, 0x401F81F4U
- #define **IOMUXC\_GPIO\_EMC\_27\_LPUART8\_RX** 0x401F8080U, 0x2U, 0x401F8404U, 0x1U, 0x401F81F4U
- #define **IOMUXC\_GPIO\_EMC\_27\_SAI1\_TX\_SYNC** 0x401F8080U, 0x3U, 0x401F8450U, 0x2U, 0x401F81F4U
- #define **IOMUXC\_GPIO\_EMC\_27\_FLEXIO1\_FLEXIO31** 0x401F8080U, 0x4U, 0, 0, 0x401F81F4U
- #define **IOMUXC\_GPIO\_EMC\_27\_GPIO2\_IO27** 0x401F8080U, 0x5U, 0, 0, 0x401F81F4U
- #define **IOMUXC\_GPIO\_EMC\_27\_SRC\_BT\_CFG09** 0x401F8080U, 0x6U, 0, 0, 0x401F81F4U
- #define **IOMUXC\_GPIO\_EMC\_28\_SEMC\_DQS** 0x401F8084U, 0x0U, 0, 0, 0x401F81F8U
- #define **IOMUXC\_GPIO\_EMC\_28\_FLEXPWM2\_PWMA03** 0x401F8084U, 0x1U, 0x401F8354U, 0x1U, 0x401F81F8U
- #define **IOMUXC\_GPIO\_EMC\_28\_XBAR1\_INOUT18** 0x401F8084U, 0x2U, 0x401F84BCU, 0x0U, 0x401F81F8U
- #define **IOMUXC\_GPIO\_EMC\_28\_SAI3\_MCLK** 0x401F8084U, 0x3U, 0x401F846CU, 0x2U, 0x401F81F8U
- #define **IOMUXC\_GPIO\_EMC\_28\_EWM\_OUT\_B** 0x401F8084U, 0x4U, 0, 0, 0x401F81F8U
- #define **IOMUXC\_GPIO\_EMC\_28\_GPIO2\_IO28** 0x401F8084U, 0x5U, 0, 0, 0x401F81F8U
- #define **IOMUXC\_GPIO\_EMC\_28\_GPT2\_CAPTURE2** 0x401F8084U, 0x6U, 0, 0, 0x401F81F8U
- #define **IOMUXC\_GPIO\_EMC\_28\_FLEXPWM1\_PWMX00** 0x401F8084U, 0x7U, 0, 0, 0x401F81F8U
- #define **IOMUXC\_GPIO\_EMC\_29\_SEMC\_CKE** 0x401F8088U, 0x0U, 0, 0, 0x401F81FCU
- #define **IOMUXC\_GPIO\_EMC\_29\_FLEXPWM2\_PWMB03** 0x401F8088U, 0x1U, 0x401F8364U, 0x1U, 0x401F81FCU
- #define **IOMUXC\_GPIO\_EMC\_29\_XBAR1\_INOUT19** 0x401F8088U, 0x2U, 0x401F84C0U, 0x0U, 0x401F81FCU
- #define **IOMUXC\_GPIO\_EMC\_29\_SAI3\_RX\_BCLK** 0x401F8088U, 0x3U, 0x401F8470U, 0x1U, 0x401F81FCU
- #define **IOMUXC\_GPIO\_EMC\_29\_WDOG2\_RST\_B\_DEB** 0x401F8088U, 0x4U, 0, 0, 0x401F81FCU
- #define **IOMUXC\_GPIO\_EMC\_29\_GPIO2\_IO29** 0x401F8088U, 0x5U, 0, 0, 0x401F81FCU
- #define **IOMUXC\_GPIO\_EMC\_29\_GPT2\_COMPARE2** 0x401F8088U, 0x6U, 0, 0, 0x401F81FCU

- #define **IOMUXC\_GPIO\_EMC\_29\_FLEXPWM1\_PWMX01** 0x401F8088U, 0x7U, 0, 0, 0x401F81FCU
- #define **IOMUXC\_GPIO\_EMC\_30\_SEMC\_CLK** 0x401F808CU, 0x0U, 0, 0, 0x401F8200U
- #define **IOMUXC\_GPIO\_EMC\_30\_FLEXPWM2\_PWMA02** 0x401F808CU, 0x1U, 0x401F8350U, 0x1U, 0x401F8200U
- #define **IOMUXC\_GPIO\_EMC\_30\_LPUART4\_CTS\_B** 0x401F808CU, 0x2U, 0x401F83E0U, 0x1U, 0x401F8200U
- #define **IOMUXC\_GPIO\_EMC\_30\_SAI3\_RX\_SYNC** 0x401F808CU, 0x3U, 0x401F8478U, 0x1U, 0x401F8200U
- #define **IOMUXC\_GPIO\_EMC\_30\_WDOG1\_RST\_B\_DEB** 0x401F808CU, 0x4U, 0, 0, 0x401F8200U
- #define **IOMUXC\_GPIO\_EMC\_30\_GPIO2\_IO30** 0x401F808CU, 0x5U, 0, 0, 0x401F8200U
- #define **IOMUXC\_GPIO\_EMC\_30\_GPT2\_COMPARE3** 0x401F808CU, 0x6U, 0, 0, 0x401F8200U
- #define **IOMUXC\_GPIO\_EMC\_30\_FLEXPWM1\_PWMX02** 0x401F808CU, 0x7U, 0, 0, 0x401F8200U
- #define **IOMUXC\_GPIO\_EMC\_31\_SEMC\_DM01** 0x401F8090U, 0x0U, 0, 0, 0x401F8204U
- #define **IOMUXC\_GPIO\_EMC\_31\_FLEXPWM2\_PWMB02** 0x401F8090U, 0x1U, 0x401F8360U, 0x1U, 0x401F8204U
- #define **IOMUXC\_GPIO\_EMC\_31\_LPUART4\_RTS\_B** 0x401F8090U, 0x2U, 0, 0, 0x401F8204U
- #define **IOMUXC\_GPIO\_EMC\_31\_SAI3\_RX\_DATA** 0x401F8090U, 0x3U, 0x401F8474U, 0x1U, 0x401F8204U
- #define **IOMUXC\_GPIO\_EMC\_31\_WDOG2\_B** 0x401F8090U, 0x4U, 0, 0, 0x401F8204U
- #define **IOMUXC\_GPIO\_EMC\_31\_GPIO2\_IO31** 0x401F8090U, 0x5U, 0, 0, 0x401F8204U
- #define **IOMUXC\_GPIO\_EMC\_31\_GPT2\_CLK** 0x401F8090U, 0x6U, 0, 0, 0x401F8204U
- #define **IOMUXC\_GPIO\_EMC\_31\_FLEXPWM1\_PWMX03** 0x401F8090U, 0x7U, 0, 0, 0x401F8204U
- #define **IOMUXC\_GPIO\_EMC\_32\_SEMC\_DATA08** 0x401F8094U, 0x0U, 0, 0, 0x401F8208U
- #define **IOMUXC\_GPIO\_EMC\_32\_QTIMER1\_TIMER0** 0x401F8094U, 0x1U, 0x401F8410U, 0x1U, 0x401F8208U
- #define **IOMUXC\_GPIO\_EMC\_32\_LPUART4\_TX** 0x401F8094U, 0x2U, 0x401F83E8U, 0x2U, 0x401F8208U
- #define **IOMUXC\_GPIO\_EMC\_32\_SAI3\_TX\_DATA** 0x401F8094U, 0x3U, 0, 0, 0x401F8208U
- #define **IOMUXC\_GPIO\_EMC\_32\_LPSPi4\_SCK** 0x401F8094U, 0x4U, 0x401F83C0U, 0x1U, 0x401F8208U
- #define **IOMUXC\_GPIO\_EMC\_32\_GPIO3\_IO00** 0x401F8094U, 0x5U, 0, 0, 0x401F8208U
- #define **IOMUXC\_GPIO\_EMC\_32\_REF\_24M\_OUT** 0x401F8094U, 0x7U, 0, 0, 0x401F8208U
- #define **IOMUXC\_GPIO\_EMC\_33\_SEMC\_DATA09** 0x401F8098U, 0x0U, 0, 0, 0x401F820CU
- #define **IOMUXC\_GPIO\_EMC\_33\_QTIMER1\_TIMER1** 0x401F8098U, 0x1U, 0x401F8414U, 0x1U, 0x401F820CU
- #define **IOMUXC\_GPIO\_EMC\_33\_LPUART4\_RX** 0x401F8098U, 0x2U, 0x401F83E4U, 0x2U, 0x401F820CU
- #define **IOMUXC\_GPIO\_EMC\_33\_SAI3\_TX\_BCLK** 0x401F8098U, 0x3U, 0x401F847CU, 0x1U, 0x401F820CU
- #define **IOMUXC\_GPIO\_EMC\_33\_LPSPi4\_PCS0** 0x401F8098U, 0x4U, 0x401F83BCU, 0x1U, 0x401F820CU
- #define **IOMUXC\_GPIO\_EMC\_33\_GPIO3\_IO01** 0x401F8098U, 0x5U, 0, 0, 0x401F820CU
- #define **IOMUXC\_GPIO\_EMC\_34\_SEMC\_DATA10** 0x401F809CU, 0x0U, 0, 0, 0x401F8210U
- #define **IOMUXC\_GPIO\_EMC\_34\_QTIMER1\_TIMER2** 0x401F809CU, 0x1U, 0x401F8418U, 0x1U, 0x401F8210U

- #define **IOMUXC\_GPIO\_EMC\_34\_LPUART7\_TX** 0x401F809CU, 0x2U, 0x401F8400U, 0x1-U, 0x401F8210U
- #define **IOMUXC\_GPIO\_EMC\_34\_SAI3\_TX\_SYNC** 0x401F809CU, 0x3U, 0x401F8480U, 0x1U, 0x401F8210U
- #define **IOMUXC\_GPIO\_EMC\_34\_LPSPi4\_SDO** 0x401F809CU, 0x4U, 0x401F83C8U, 0x1U, 0x401F8210U
- #define **IOMUXC\_GPIO\_EMC\_34\_GPIO3\_IO02** 0x401F809CU, 0x5U, 0, 0, 0x401F8210U
- #define **IOMUXC\_GPIO\_EMC\_34\_ENET\_CRs** 0x401F809CU, 0x6U, 0, 0, 0x401F8210U
- #define **IOMUXC\_GPIO\_EMC\_35\_SEMC\_DATA11** 0x401F80A0U, 0x0U, 0, 0, 0x401F8214U
- #define **IOMUXC\_GPIO\_EMC\_35\_QTIMER1\_TIMER3** 0x401F80A0U, 0x1U, 0x401F841C-U, 0x1U, 0x401F8214U
- #define **IOMUXC\_GPIO\_EMC\_35\_LPUART7\_RX** 0x401F80A0U, 0x2U, 0x401F83FCU, 0x1-U, 0x401F8214U
- #define **IOMUXC\_GPIO\_EMC\_35\_USDHC2\_WP** 0x401F80A0U, 0x3U, 0x401F849CU, 0x1U, 0x401F8214U
- #define **IOMUXC\_GPIO\_EMC\_35\_LPSPi4\_SDI** 0x401F80A0U, 0x4U, 0x401F83C4U, 0x1U, 0x401F8214U
- #define **IOMUXC\_GPIO\_EMC\_35\_GPIO3\_IO03** 0x401F80A0U, 0x5U, 0, 0, 0x401F8214U
- #define **IOMUXC\_GPIO\_EMC\_35\_ENET\_COL** 0x401F80A0U, 0x6U, 0, 0, 0x401F8214U
- #define **IOMUXC\_GPIO\_EMC\_36\_SEMC\_DATA12** 0x401F80A4U, 0x0U, 0, 0, 0x401F8218U
- #define **IOMUXC\_GPIO\_EMC\_36\_FLEXPWM2\_PWMA01** 0x401F80A4U, 0x1U, 0x401-F834CU, 0x1U, 0x401F8218U
- #define **IOMUXC\_GPIO\_EMC\_36\_LPUART5\_CTS\_B** 0x401F80A4U, 0x2U, 0, 0, 0x401-F8218U
- #define **IOMUXC\_GPIO\_EMC\_36\_CCM\_PMIC\_RDY** 0x401F80A4U, 0x3U, 0x401F8300U, 0x3U, 0x401F8218U
- #define **IOMUXC\_GPIO\_EMC\_36\_LPSPi4\_PCS1** 0x401F80A4U, 0x4U, 0, 0, 0x401F8218U
- #define **IOMUXC\_GPIO\_EMC\_36\_GPIO3\_IO04** 0x401F80A4U, 0x5U, 0, 0, 0x401F8218U
- #define **IOMUXC\_GPIO\_EMC\_36\_ENET\_RX\_CLK** 0x401F80A4U, 0x6U, 0, 0, 0x401F8218U
- #define **IOMUXC\_GPIO\_EMC\_36\_USDHC1\_WP** 0x401F80A4U, 0x7U, 0x401F8494U, 0x4U, 0x401F8218U
- #define **IOMUXC\_GPIO\_EMC\_37\_SEMC\_DATA13** 0x401F80A8U, 0x0U, 0, 0, 0x401F821CU
- #define **IOMUXC\_GPIO\_EMC\_37\_FLEXPWM2\_PWMB01** 0x401F80A8U, 0x1U, 0x401-F835CU, 0x1U, 0x401F821CU
- #define **IOMUXC\_GPIO\_EMC\_37\_LPUART5\_RTS\_B** 0x401F80A8U, 0x2U, 0, 0, 0x401F821-CU
- #define **IOMUXC\_GPIO\_EMC\_37\_MQS\_RIGHT** 0x401F80A8U, 0x3U, 0, 0, 0x401F821CU
- #define **IOMUXC\_GPIO\_EMC\_37\_LPSPi4\_PCS2** 0x401F80A8U, 0x4U, 0, 0, 0x401F821CU
- #define **IOMUXC\_GPIO\_EMC\_37\_GPIO3\_IO05** 0x401F80A8U, 0x5U, 0, 0, 0x401F821CU
- #define **IOMUXC\_GPIO\_EMC\_37\_ENET\_RDATA03** 0x401F80A8U, 0x6U, 0, 0, 0x401F821-CU
- #define **IOMUXC\_GPIO\_EMC\_37\_USDHC1\_VSELECT** 0x401F80A8U, 0x7U, 0, 0, 0x401-F821CU
- #define **IOMUXC\_GPIO\_EMC\_38\_SEMC\_DATA14** 0x401F80ACU, 0x0U, 0, 0, 0x401F8220U
- #define **IOMUXC\_GPIO\_EMC\_38\_FLEXPWM2\_PWMA00** 0x401F80ACU, 0x1U, 0x401-F8348U, 0x1U, 0x401F8220U
- #define **IOMUXC\_GPIO\_EMC\_38\_LPUART5\_TX** 0x401F80ACU, 0x2U, 0x401F83F0U, 0x1-U, 0x401F8220U
- #define **IOMUXC\_GPIO\_EMC\_38\_MQS\_LEFT** 0x401F80ACU, 0x3U, 0, 0, 0x401F8220U
- #define **IOMUXC\_GPIO\_EMC\_38\_LPSPi4\_PCS3** 0x401F80ACU, 0x4U, 0, 0, 0x401F8220U
- #define **IOMUXC\_GPIO\_EMC\_38\_GPIO3\_IO06** 0x401F80ACU, 0x5U, 0, 0, 0x401F8220U
- #define **IOMUXC\_GPIO\_EMC\_38\_ENET\_RDATA02** 0x401F80ACU, 0x6U, 0, 0, 0x401-



- F8220U
- #define **IOMUXC\_GPIO\_EMC\_38\_USDHC1\_CD\_B** 0x401F80ACU, 0x7U, 0x401F8490U, 0x3U, 0x401F8220U
- #define **IOMUXC\_GPIO\_EMC\_39\_SEMC\_DATA15** 0x401F80B0U, 0x0U, 0, 0, 0x401F8224U
- #define **IOMUXC\_GPIO\_EMC\_39\_FLEXPWM2\_PWMB00** 0x401F80B0U, 0x1U, 0x401F8358U, 0x1U, 0x401F8224U
- #define **IOMUXC\_GPIO\_EMC\_39\_LPUART5\_RX** 0x401F80B0U, 0x2U, 0x401F83ECU, 0x1U, 0x401F8224U
- #define **IOMUXC\_GPIO\_EMC\_39\_USB\_OTG1\_OC** 0x401F80B0U, 0x3U, 0x401F848CU, 0x2U, 0x401F8224U
- #define **IOMUXC\_GPIO\_EMC\_39\_WDOG1\_B** 0x401F80B0U, 0x4U, 0, 0, 0x401F8224U
- #define **IOMUXC\_GPIO\_EMC\_39\_GPIO3\_IO07** 0x401F80B0U, 0x5U, 0, 0, 0x401F8224U
- #define **IOMUXC\_GPIO\_EMC\_39\_ENET\_TX\_ER** 0x401F80B0U, 0x6U, 0, 0, 0x401F8224U
- #define **IOMUXC\_GPIO\_EMC\_39\_GPT1\_CLK** 0x401F80B0U, 0x7U, 0, 0, 0x401F8224U
- #define **IOMUXC\_GPIO\_EMC\_40\_SEMC\_CSX00** 0x401F80B4U, 0x0U, 0, 0, 0x401F8228U
- #define **IOMUXC\_GPIO\_EMC\_40\_XBAR1\_INOUT18** 0x401F80B4U, 0x1U, 0x401F84BCU, 0x1U, 0x401F8228U
- #define **IOMUXC\_GPIO\_EMC\_40\_SPDIF\_OUT** 0x401F80B4U, 0x2U, 0, 0, 0x401F8228U
- #define **IOMUXC\_GPIO\_EMC\_40\_USB\_OTG1\_ID** 0x401F80B4U, 0x3U, 0x401F82FCU, 0x2U, 0x401F8228U
- #define **IOMUXC\_GPIO\_EMC\_40\_ENET\_MDIO** 0x401F80B4U, 0x4U, 0x401F8308U, 0x2U, 0x401F8228U
- #define **IOMUXC\_GPIO\_EMC\_40\_GPIO3\_IO08** 0x401F80B4U, 0x5U, 0, 0, 0x401F8228U
- #define **IOMUXC\_GPIO\_EMC\_40\_ENET\_TDATA03** 0x401F80B4U, 0x6U, 0, 0, 0x401F8228U
- #define **IOMUXC\_GPIO\_EMC\_40\_GPT1\_COMPARE3** 0x401F80B4U, 0x7U, 0, 0, 0x401F8228U
- #define **IOMUXC\_GPIO\_EMC\_41\_SEMC\_READY** 0x401F80B8U, 0x0U, 0x401F8484U, 0x1U, 0x401F822CU
- #define **IOMUXC\_GPIO\_EMC\_41\_XBAR1\_INOUT19** 0x401F80B8U, 0x1U, 0x401F84C0U, 0x1U, 0x401F822CU
- #define **IOMUXC\_GPIO\_EMC\_41\_SPDIF\_IN** 0x401F80B8U, 0x2U, 0x401F8488U, 0x1U, 0x401F822CU
- #define **IOMUXC\_GPIO\_EMC\_41\_USB\_OTG1\_PWR** 0x401F80B8U, 0x3U, 0, 0, 0x401F822CU
- #define **IOMUXC\_GPIO\_EMC\_41\_ENET\_MDC** 0x401F80B8U, 0x4U, 0, 0, 0x401F822CU
- #define **IOMUXC\_GPIO\_EMC\_41\_GPIO3\_IO09** 0x401F80B8U, 0x5U, 0, 0, 0x401F822CU
- #define **IOMUXC\_GPIO\_EMC\_41\_ENET\_TDATA02** 0x401F80B8U, 0x6U, 0, 0, 0x401F822CU
- #define **IOMUXC\_GPIO\_EMC\_41\_GPT1\_COMPARE2** 0x401F80B8U, 0x7U, 0, 0, 0x401F822CU
- #define **IOMUXC\_GPIO\_AD\_B0\_00\_JTAG\_TMS** 0x401F80BCU, 0x0U, 0, 0, 0x401F8230U
- #define **IOMUXC\_GPIO\_AD\_B0\_00\_GPIO1\_IO00** 0x401F80BCU, 0x5U, 0, 0, 0x401F8230U
- #define **IOMUXC\_GPIO\_AD\_B0\_00\_GPT1\_COMPARE1** 0x401F80BCU, 0x7U, 0, 0, 0x401F8230U
- #define **IOMUXC\_GPIO\_AD\_B0\_01\_JTAG\_TCK** 0x401F80C0U, 0x0U, 0, 0, 0x401F8234U
- #define **IOMUXC\_GPIO\_AD\_B0\_01\_GPIO1\_IO01** 0x401F80C0U, 0x5U, 0, 0, 0x401F8234U
- #define **IOMUXC\_GPIO\_AD\_B0\_01\_GPT1\_CAPTURE2** 0x401F80C0U, 0x7U, 0, 0, 0x401F8234U
- #define **IOMUXC\_GPIO\_AD\_B0\_02\_JTAG\_MOD** 0x401F80C4U, 0x0U, 0, 0, 0x401F8238U
- #define **IOMUXC\_GPIO\_AD\_B0\_02\_GPIO1\_IO02** 0x401F80C4U, 0x5U, 0, 0, 0x401F8238U
- #define **IOMUXC\_GPIO\_AD\_B0\_02\_GPT1\_CAPTURE1** 0x401F80C4U, 0x7U, 0, 0, 0x401F8238U

```

F8238U
• #define IOMUXC_GPIO_AD_B0_03_JTAG_TDI 0x401F80C8U, 0x0U, 0, 0, 0x401F823CU
• #define IOMUXC_GPIO_AD_B0_03_USDHC2_CD_B 0x401F80C8U, 0x1U, 0x401F8498U,
 0x1U, 0x401F823CU
• #define IOMUXC_GPIO_AD_B0_03_WDOG1_B 0x401F80C8U, 0x2U, 0, 0, 0x401F823CU
• #define IOMUXC_GPIO_AD_B0_03_SAI1_MCLK 0x401F80C8U, 0x3U, 0x401F8430U, 0x1-
 U, 0x401F823CU
• #define IOMUXC_GPIO_AD_B0_03_USDHC1_WP 0x401F80C8U, 0x4U, 0x401F8494U, 0x0-
 U, 0x401F823CU
• #define IOMUXC_GPIO_AD_B0_03_GPIO1_IO03 0x401F80C8U, 0x5U, 0, 0, 0x401F823CU
• #define IOMUXC_GPIO_AD_B0_03_USB_OTG1_OC 0x401F80C8U, 0x6U, 0x401F848CU,
 0x0U, 0x401F823CU
• #define IOMUXC_GPIO_AD_B0_03_CCM_PMIC_RDY 0x401F80C8U, 0x7U, 0x401F8300U,
 0x2U, 0x401F823CU
• #define IOMUXC_GPIO_AD_B0_04_JTAG_TDO 0x401F80CCU, 0x0U, 0, 0, 0x401F8240U
• #define IOMUXC_GPIO_AD_B0_04_FLEXCAN1_TX 0x401F80CCU, 0x1U, 0, 0, 0x401-
 F8240U
• #define IOMUXC_GPIO_AD_B0_04_USDHC1_WP 0x401F80CCU, 0x2U, 0x401F8494U, 0x1-
 U, 0x401F8240U
• #define IOMUXC_GPIO_AD_B0_04_QTIMER2_TIMER0 0x401F80CCU, 0x3U, 0x401-
 F8420U, 0x1U, 0x401F8240U
• #define IOMUXC_GPIO_AD_B0_04_ENET_MDIO 0x401F80CCU, 0x4U, 0x401F8308U, 0x1-
 U, 0x401F8240U
• #define IOMUXC_GPIO_AD_B0_04_GPIO1_IO04 0x401F80CCU, 0x5U, 0, 0, 0x401F8240U
• #define IOMUXC_GPIO_AD_B0_04_USB_OTG1_PWR 0x401F80CCU, 0x6U, 0, 0, 0x401-
 F8240U
• #define IOMUXC_GPIO_AD_B0_04_EWM_OUT_B 0x401F80CCU, 0x7U, 0, 0, 0x401F8240U
• #define IOMUXC_GPIO_AD_B0_05_JTAG_TRSTB 0x401F80D0U, 0x0U, 0, 0, 0x401F8244U
• #define IOMUXC_GPIO_AD_B0_05_FLEXCAN1_RX 0x401F80D0U, 0x1U, 0x401F8320U,
 0x2U, 0x401F8244U
• #define IOMUXC_GPIO_AD_B0_05_USDHC1_CD_B 0x401F80D0U, 0x2U, 0x401F8490U,
 0x1U, 0x401F8244U
• #define IOMUXC_GPIO_AD_B0_05_QTIMER2_TIMER1 0x401F80D0U, 0x3U, 0x401-
 F8424U, 0x1U, 0x401F8244U
• #define IOMUXC_GPIO_AD_B0_05_ENET_MDC 0x401F80D0U, 0x4U, 0, 0, 0x401F8244U
• #define IOMUXC_GPIO_AD_B0_05_GPIO1_IO05 0x401F80D0U, 0x5U, 0, 0, 0x401F8244U
• #define IOMUXC_GPIO_AD_B0_05_USB_OTG1_ID 0x401F80D0U, 0x6U, 0x401F82FCU,
 0x0U, 0x401F8244U
• #define IOMUXC_GPIO_AD_B0_05_NMI_GLUE_NMI 0x401F80D0U, 0x7U, 0x401F840CU,
 0x0U, 0x401F8244U
• #define IOMUXC_GPIO_AD_B0_06_PIT_TRIGGER00 0x401F80D4U, 0x0U, 0, 0, 0x401-
 F8248U
• #define IOMUXC_GPIO_AD_B0_06_MQS_RIGHT 0x401F80D4U, 0x1U, 0, 0, 0x401F8248U
• #define IOMUXC_GPIO_AD_B0_06_LPUART1_TX 0x401F80D4U, 0x2U, 0, 0, 0x401F8248U
• #define IOMUXC_GPIO_AD_B0_06_QTIMER2_TIMER2 0x401F80D4U, 0x3U, 0x401-
 F8428U, 0x1U, 0x401F8248U
• #define IOMUXC_GPIO_AD_B0_06_FLEXPWM2_PWMA03 0x401F80D4U, 0x4U, 0x401-
 F8354U, 0x0U, 0x401F8248U
• #define IOMUXC_GPIO_AD_B0_06_GPIO1_IO06 0x401F80D4U, 0x5U, 0, 0, 0x401F8248U
• #define IOMUXC_GPIO_AD_B0_06_REF_32K_OUT 0x401F80D4U, 0x6U, 0, 0, 0x401F8248-
 U

```

- #define **IOMUXC\_GPIO\_AD\_B0\_07\_PIT\_TRIGGER01** 0x401F80D8U, 0x0U, 0, 0, 0x401F824CU
- #define **IOMUXC\_GPIO\_AD\_B0\_07\_MQS\_LEFT** 0x401F80D8U, 0x1U, 0, 0, 0x401F824CU
- #define **IOMUXC\_GPIO\_AD\_B0\_07\_LPUART1\_RX** 0x401F80D8U, 0x2U, 0, 0, 0x401F824CU
- #define **IOMUXC\_GPIO\_AD\_B0\_07\_QTIMER2\_TIMER3** 0x401F80D8U, 0x3U, 0x401F842CU, 0x1U, 0x401F824CU
- #define **IOMUXC\_GPIO\_AD\_B0\_07\_FLEXPWM2\_PWMB03** 0x401F80D8U, 0x4U, 0x401F8364U, 0x0U, 0x401F824CU
- #define **IOMUXC\_GPIO\_AD\_B0\_07\_GPIO1\_IO07** 0x401F80D8U, 0x5U, 0, 0, 0x401F824CU
- #define **IOMUXC\_GPIO\_AD\_B0\_07\_REF\_24M\_OUT** 0x401F80D8U, 0x6U, 0, 0, 0x401F824CU
- #define **IOMUXC\_GPIO\_AD\_B0\_08\_ENET\_TX\_CLK** 0x401F80DCU, 0x0U, 0x401F831CU, 0x1U, 0x401F8250U
- #define **IOMUXC\_GPIO\_AD\_B0\_08\_LPI2C3\_SCL** 0x401F80DCU, 0x1U, 0x401F838CU, 0x1U, 0x401F8250U
- #define **IOMUXC\_GPIO\_AD\_B0\_08\_LPUART1\_CTS\_B** 0x401F80DCU, 0x2U, 0, 0, 0x401F8250U
- #define **IOMUXC\_GPIO\_AD\_B0\_08\_KPP\_COL00** 0x401F80DCU, 0x3U, 0, 0, 0x401F8250U
- #define **IOMUXC\_GPIO\_AD\_B0\_08\_ENET\_REF\_CLK1** 0x401F80DCU, 0x4U, 0x401F8304U, 0x1U, 0x401F8250U
- #define **IOMUXC\_GPIO\_AD\_B0\_08\_GPIO1\_IO08** 0x401F80DCU, 0x5U, 0, 0, 0x401F8250U
- #define **IOMUXC\_GPIO\_AD\_B0\_08\_ARM\_CM7\_TXEV** 0x401F80DCU, 0x6U, 0, 0, 0x401F8250U
- #define **IOMUXC\_GPIO\_AD\_B0\_09\_ENET\_RDATA01** 0x401F80E0U, 0x0U, 0x401F8310U, 0x1U, 0x401F8254U
- #define **IOMUXC\_GPIO\_AD\_B0\_09\_LPI2C3\_SDA** 0x401F80E0U, 0x1U, 0x401F8390U, 0x1U, 0x401F8254U
- #define **IOMUXC\_GPIO\_AD\_B0\_09\_LPUART1\_RTS\_B** 0x401F80E0U, 0x2U, 0, 0, 0x401F8254U
- #define **IOMUXC\_GPIO\_AD\_B0\_09\_KPP\_ROW00** 0x401F80E0U, 0x3U, 0, 0, 0x401F8254U
- #define **IOMUXC\_GPIO\_AD\_B0\_09\_GPIO1\_IO09** 0x401F80E0U, 0x5U, 0, 0, 0x401F8254U
- #define **IOMUXC\_GPIO\_AD\_B0\_09\_ARM\_CM7\_RXEV** 0x401F80E0U, 0x6U, 0, 0, 0x401F8254U
- #define **IOMUXC\_GPIO\_AD\_B0\_10\_ENET\_RDATA00** 0x401F80E4U, 0x0U, 0x401F830CU, 0x1U, 0x401F8258U
- #define **IOMUXC\_GPIO\_AD\_B0\_10\_LPSPI1\_SCK** 0x401F80E4U, 0x1U, 0x401F83A0U, 0x1U, 0x401F8258U
- #define **IOMUXC\_GPIO\_AD\_B0\_10\_LPUART5\_TX** 0x401F80E4U, 0x2U, 0x401F83F0U, 0x0U, 0x401F8258U
- #define **IOMUXC\_GPIO\_AD\_B0\_10\_KPP\_COL01** 0x401F80E4U, 0x3U, 0, 0, 0x401F8258U
- #define **IOMUXC\_GPIO\_AD\_B0\_10\_FLEXPWM2\_PWMA02** 0x401F80E4U, 0x4U, 0x401F8350U, 0x0U, 0x401F8258U
- #define **IOMUXC\_GPIO\_AD\_B0\_10\_GPIO1\_IO10** 0x401F80E4U, 0x5U, 0, 0, 0x401F8258U
- #define **IOMUXC\_GPIO\_AD\_B0\_10\_ARM\_CM7\_TRACE\_CLK** 0x401F80E4U, 0x6U, 0, 0, 0x401F8258U
- #define **IOMUXC\_GPIO\_AD\_B0\_11\_ENET\_RX\_EN** 0x401F80E8U, 0x0U, 0x401F8314U, 0x1U, 0x401F825CU
- #define **IOMUXC\_GPIO\_AD\_B0\_11\_LPSPI1\_PCS0** 0x401F80E8U, 0x1U, 0x401F839CU, 0x1U, 0x401F825CU
- #define **IOMUXC\_GPIO\_AD\_B0\_11\_LPUART5\_RX** 0x401F80E8U, 0x2U, 0x401F83ECU,

- 0x0U, 0x401F825CU
- #define **IOMUXC\_GPIO\_AD\_B0\_11\_KPP\_ROW01** 0x401F80E8U, 0x3U, 0, 0, 0x401F825CU
- #define **IOMUXC\_GPIO\_AD\_B0\_11\_FLEXPWM2\_PWMB02** 0x401F80E8U, 0x4U, 0x401-F8360U, 0x0U, 0x401F825CU
- #define **IOMUXC\_GPIO\_AD\_B0\_11\_GPIO1\_IO11** 0x401F80E8U, 0x5U, 0, 0, 0x401F825CU
- #define **IOMUXC\_GPIO\_AD\_B0\_11\_ARM\_CM7\_TRACE\_SWO** 0x401F80E8U, 0x6U, 0, 0, 0x401F825CU
- #define **IOMUXC\_GPIO\_AD\_B0\_12\_ENET\_RX\_ER** 0x401F80ECU, 0x0U, 0x401F8318U, 0x1U, 0x401F8260U
- #define **IOMUXC\_GPIO\_AD\_B0\_12\_LPSP1\_SDO** 0x401F80ECU, 0x1U, 0x401F83A8U, 0x1-U, 0x401F8260U
- #define **IOMUXC\_GPIO\_AD\_B0\_12\_LPUART3\_CTS\_B** 0x401F80ECU, 0x2U, 0, 0, 0x401-F8260U
- #define **IOMUXC\_GPIO\_AD\_B0\_12\_KPP\_COL02** 0x401F80ECU, 0x3U, 0, 0, 0x401F8260U
- #define **IOMUXC\_GPIO\_AD\_B0\_12\_FLEXPWM2\_PWMA01** 0x401F80ECU, 0x4U, 0x401-F834CU, 0x0U, 0x401F8260U
- #define **IOMUXC\_GPIO\_AD\_B0\_12\_GPIO1\_IO12** 0x401F80ECU, 0x5U, 0, 0, 0x401F8260U
- #define **IOMUXC\_GPIO\_AD\_B0\_12\_ARM\_CM7\_TRACE00** 0x401F80ECU, 0x6U, 0, 0, 0x401F8260U
- #define **IOMUXC\_GPIO\_AD\_B0\_12\_SNV5\_HP\_VIO\_5\_CTL** 0x401F80ECU, 0x7U, 0, 0, 0x401F8260U
- #define **IOMUXC\_GPIO\_AD\_B0\_13\_ENET\_TX\_EN** 0x401F80F0U, 0x0U, 0, 0, 0x401F8264U
- #define **IOMUXC\_GPIO\_AD\_B0\_13\_LPSP1\_SDI** 0x401F80F0U, 0x1U, 0x401F83A4U, 0x1U, 0x401F8264U
- #define **IOMUXC\_GPIO\_AD\_B0\_13\_LPUART3\_RTS\_B** 0x401F80F0U, 0x2U, 0, 0, 0x401-F8264U
- #define **IOMUXC\_GPIO\_AD\_B0\_13\_KPP\_ROW02** 0x401F80F0U, 0x3U, 0, 0, 0x401F8264U
- #define **IOMUXC\_GPIO\_AD\_B0\_13\_FLEXPWM2\_PWMB01** 0x401F80F0U, 0x4U, 0x401-F835CU, 0x0U, 0x401F8264U
- #define **IOMUXC\_GPIO\_AD\_B0\_13\_GPIO1\_IO13** 0x401F80F0U, 0x5U, 0, 0, 0x401F8264U
- #define **IOMUXC\_GPIO\_AD\_B0\_13\_ARM\_CM7\_TRACE01** 0x401F80F0U, 0x6U, 0, 0, 0x401F8264U
- #define **IOMUXC\_GPIO\_AD\_B0\_13\_SNV5\_HP\_VIO\_5\_B** 0x401F80F0U, 0x7U, 0, 0, 0x401-F8264U
- #define **IOMUXC\_GPIO\_AD\_B0\_14\_ENET\_TDATA00** 0x401F80F4U, 0x0U, 0, 0, 0x401-F8268U
- #define **IOMUXC\_GPIO\_AD\_B0\_14\_FLEXPWM2\_TX** 0x401F80F4U, 0x1U, 0, 0, 0x401-F8268U
- #define **IOMUXC\_GPIO\_AD\_B0\_14\_LPUART3\_TX** 0x401F80F4U, 0x2U, 0x401F83DCU, 0x1U, 0x401F8268U
- #define **IOMUXC\_GPIO\_AD\_B0\_14\_KPP\_COL03** 0x401F80F4U, 0x3U, 0, 0, 0x401F8268U
- #define **IOMUXC\_GPIO\_AD\_B0\_14\_FLEXPWM2\_PWMA00** 0x401F80F4U, 0x4U, 0x401-F8348U, 0x0U, 0x401F8268U
- #define **IOMUXC\_GPIO\_AD\_B0\_14\_GPIO1\_IO14** 0x401F80F4U, 0x5U, 0, 0, 0x401F8268U
- #define **IOMUXC\_GPIO\_AD\_B0\_14\_ARM\_CM7\_TRACE02** 0x401F80F4U, 0x6U, 0, 0, 0x401F8268U
- #define **IOMUXC\_GPIO\_AD\_B0\_14\_WDOG1\_ANY** 0x401F80F4U, 0x7U, 0, 0, 0x401F8268U
- #define **IOMUXC\_GPIO\_AD\_B0\_15\_ENET\_TDATA01** 0x401F80F8U, 0x0U, 0, 0, 0x401-F826CU
- #define **IOMUXC\_GPIO\_AD\_B0\_15\_FLEXPWM2\_RX** 0x401F80F8U, 0x1U, 0x401F8324U, 0x2U, 0x401F826CU

- #define **IOMUXC\_GPIO\_AD\_B0\_15\_LPUART3\_RX** 0x401F80F8U, 0x2U, 0x401F83D8U, 0x1U, 0x401F826CU
- #define **IOMUXC\_GPIO\_AD\_B0\_15\_KPP\_ROW03** 0x401F80F8U, 0x3U, 0, 0, 0x401F826CU
- #define **IOMUXC\_GPIO\_AD\_B0\_15\_FLEXPWM2\_PWMB00** 0x401F80F8U, 0x4U, 0x401F8358U, 0x0U, 0x401F826CU
- #define **IOMUXC\_GPIO\_AD\_B0\_15\_GPIO1\_IO15** 0x401F80F8U, 0x5U, 0, 0, 0x401F826CU
- #define **IOMUXC\_GPIO\_AD\_B0\_15\_ARM\_CM7\_TRACE03** 0x401F80F8U, 0x6U, 0, 0, 0x401F826CU
- #define **IOMUXC\_GPIO\_AD\_B1\_00\_SEMC\_READY** 0x401F80FCU, 0x0U, 0x401F8484U, 0x0U, 0x401F8270U
- #define **IOMUXC\_GPIO\_AD\_B1\_00\_FLEXSPI\_A\_DATA03** 0x401F80FCU, 0x1U, 0x401F8374U, 0x1U, 0x401F8270U
- #define **IOMUXC\_GPIO\_AD\_B1\_00\_FLEXCAN2\_TX** 0x401F80FCU, 0x2U, 0, 0, 0x401F8270U
- #define **IOMUXC\_GPIO\_AD\_B1\_00\_SAI1\_MCLK** 0x401F80FCU, 0x3U, 0x401F8430U, 0x2U, 0x401F8270U
- #define **IOMUXC\_GPIO\_AD\_B1\_00\_FLEXIO1\_FLEXIO15** 0x401F80FCU, 0x4U, 0, 0, 0x401F8270U
- #define **IOMUXC\_GPIO\_AD\_B1\_00\_GPIO1\_IO16** 0x401F80FCU, 0x5U, 0, 0, 0x401F8270U
- #define **IOMUXC\_GPIO\_AD\_B1\_00\_ENET\_1588\_EVENT2\_OUT** 0x401F80FCU, 0x6U, 0, 0, 0x401F8270U
- #define **IOMUXC\_GPIO\_AD\_B1\_00\_KPP\_COL04** 0x401F80FCU, 0x7U, 0, 0, 0x401F8270U
- #define **IOMUXC\_GPIO\_AD\_B1\_01\_SEMC\_CSX00** 0x401F8100U, 0x0U, 0, 0, 0x401F8274U
- #define **IOMUXC\_GPIO\_AD\_B1\_01\_FLEXSPI\_A\_SCLK** 0x401F8100U, 0x1U, 0x401F8378U, 0x1U, 0x401F8274U
- #define **IOMUXC\_GPIO\_AD\_B1\_01\_FLEXCAN2\_RX** 0x401F8100U, 0x2U, 0x401F8324U, 0x3U, 0x401F8274U
- #define **IOMUXC\_GPIO\_AD\_B1\_01\_SAI1\_TX\_BCLK** 0x401F8100U, 0x3U, 0x401F844CU, 0x1U, 0x401F8274U
- #define **IOMUXC\_GPIO\_AD\_B1\_01\_FLEXIO1\_FLEXIO14** 0x401F8100U, 0x4U, 0, 0, 0x401F8274U
- #define **IOMUXC\_GPIO\_AD\_B1\_01\_GPIO1\_IO17** 0x401F8100U, 0x5U, 0, 0, 0x401F8274U
- #define **IOMUXC\_GPIO\_AD\_B1\_01\_ENET\_1588\_EVENT2\_IN** 0x401F8100U, 0x6U, 0, 0, 0x401F8274U
- #define **IOMUXC\_GPIO\_AD\_B1\_01\_KPP\_ROW04** 0x401F8100U, 0x7U, 0, 0, 0x401F8274U
- #define **IOMUXC\_GPIO\_AD\_B1\_02\_SEMC\_CSX01** 0x401F8104U, 0x0U, 0, 0, 0x401F8278U
- #define **IOMUXC\_GPIO\_AD\_B1\_02\_FLEXSPI\_A\_DATA00** 0x401F8104U, 0x1U, 0x401F8368U, 0x1U, 0x401F8278U
- #define **IOMUXC\_GPIO\_AD\_B1\_02\_LPSPi4\_SCK** 0x401F8104U, 0x2U, 0x401F83C0U, 0x0U, 0x401F8278U
- #define **IOMUXC\_GPIO\_AD\_B1\_02\_SAI1\_TX\_SYNC** 0x401F8104U, 0x3U, 0x401F8450U, 0x1U, 0x401F8278U
- #define **IOMUXC\_GPIO\_AD\_B1\_02\_FLEXIO1\_FLEXIO13** 0x401F8104U, 0x4U, 0, 0, 0x401F8278U
- #define **IOMUXC\_GPIO\_AD\_B1\_02\_GPIO1\_IO18** 0x401F8104U, 0x5U, 0, 0, 0x401F8278U
- #define **IOMUXC\_GPIO\_AD\_B1\_02\_ENET\_1588\_EVENT3\_OUT** 0x401F8104U, 0x6U, 0, 0, 0x401F8278U
- #define **IOMUXC\_GPIO\_AD\_B1\_02\_KPP\_COL05** 0x401F8104U, 0x7U, 0, 0, 0x401F8278U
- #define **IOMUXC\_GPIO\_AD\_B1\_03\_SEMC\_CSX02** 0x401F8108U, 0x0U, 0, 0, 0x401F827CU
- #define **IOMUXC\_GPIO\_AD\_B1\_03\_FLEXSPI\_A\_DATA02** 0x401F8108U, 0x1U, 0x401F8370U, 0x1U, 0x401F827CU

- #define **IOMUXC\_GPIO\_AD\_B1\_03\_LPSP14\_PCS0** 0x401F8108U, 0x2U, 0x401F83BCU, 0x0U, 0x401F827CU
- #define **IOMUXC\_GPIO\_AD\_B1\_03\_SAI1\_TX\_DATA00** 0x401F8108U, 0x3U, 0, 0, 0x401F827CU
- #define **IOMUXC\_GPIO\_AD\_B1\_03\_FLEXIO1\_FLEXIO12** 0x401F8108U, 0x4U, 0, 0, 0x401F827CU
- #define **IOMUXC\_GPIO\_AD\_B1\_03\_GPIO1\_IO19** 0x401F8108U, 0x5U, 0, 0, 0x401F827CU
- #define **IOMUXC\_GPIO\_AD\_B1\_03\_ENET\_1588\_EVENT3\_IN** 0x401F8108U, 0x6U, 0, 0, 0x401F827CU
- #define **IOMUXC\_GPIO\_AD\_B1\_03\_KPP\_ROW05** 0x401F8108U, 0x7U, 0, 0, 0x401F827CU
- #define **IOMUXC\_GPIO\_AD\_B1\_04\_SEMC\_CSX03** 0x401F810CU, 0x0U, 0, 0, 0x401F8280U
- #define **IOMUXC\_GPIO\_AD\_B1\_04\_FLEXSPI\_A\_DATA01** 0x401F810CU, 0x1U, 0x401F836CU, 0x1U, 0x401F8280U
- #define **IOMUXC\_GPIO\_AD\_B1\_04\_LPSP14\_SDO** 0x401F810CU, 0x2U, 0x401F83C8U, 0x0U, 0x401F8280U
- #define **IOMUXC\_GPIO\_AD\_B1\_04\_SAI1\_RX\_SYNC** 0x401F810CU, 0x3U, 0x401F8448U, 0x0U, 0x401F8280U
- #define **IOMUXC\_GPIO\_AD\_B1\_04\_FLEXIO1\_FLEXIO11** 0x401F810CU, 0x4U, 0, 0, 0x401F8280U
- #define **IOMUXC\_GPIO\_AD\_B1\_04\_GPIO1\_IO20** 0x401F810CU, 0x5U, 0, 0, 0x401F8280U
- #define **IOMUXC\_GPIO\_AD\_B1\_04\_LPSP11\_PCS1** 0x401F810CU, 0x6U, 0, 0, 0x401F8280U
- #define **IOMUXC\_GPIO\_AD\_B1\_04\_KPP\_COL06** 0x401F810CU, 0x7U, 0, 0, 0x401F8280U
- #define **IOMUXC\_GPIO\_AD\_B1\_05\_USDHC1\_WP** 0x401F8110U, 0x0U, 0x401F8494U, 0x2U, 0x401F8284U
- #define **IOMUXC\_GPIO\_AD\_B1\_05\_FLEXSPI\_A\_SS0\_B** 0x401F8110U, 0x1U, 0, 0, 0x401F8284U
- #define **IOMUXC\_GPIO\_AD\_B1\_05\_LPSP14\_SDI** 0x401F8110U, 0x2U, 0x401F83C4U, 0x0U, 0x401F8284U
- #define **IOMUXC\_GPIO\_AD\_B1\_05\_SAI1\_RX\_DATA00** 0x401F8110U, 0x3U, 0x401F8438U, 0x1U, 0x401F8284U
- #define **IOMUXC\_GPIO\_AD\_B1\_05\_FLEXIO1\_FLEXIO10** 0x401F8110U, 0x4U, 0, 0, 0x401F8284U
- #define **IOMUXC\_GPIO\_AD\_B1\_05\_GPIO1\_IO21** 0x401F8110U, 0x5U, 0, 0, 0x401F8284U
- #define **IOMUXC\_GPIO\_AD\_B1\_05\_LPSP11\_PCS2** 0x401F8110U, 0x6U, 0, 0, 0x401F8284U
- #define **IOMUXC\_GPIO\_AD\_B1\_05\_KPP\_ROW06** 0x401F8110U, 0x7U, 0, 0, 0x401F8284U
- #define **IOMUXC\_GPIO\_AD\_B1\_06\_USDHC1\_RESET\_B** 0x401F8114U, 0x0U, 0, 0, 0x401F8288U
- #define **IOMUXC\_GPIO\_AD\_B1\_06\_FLEXPWM1\_PWMA00** 0x401F8114U, 0x1U, 0x401F8328U, 0x0U, 0x401F8288U
- #define **IOMUXC\_GPIO\_AD\_B1\_06\_LPUART2\_CTS\_B** 0x401F8114U, 0x2U, 0x401F83CCU, 0x0U, 0x401F8288U
- #define **IOMUXC\_GPIO\_AD\_B1\_06\_SAI1\_RX\_BCLK** 0x401F8114U, 0x3U, 0x401F8434U, 0x0U, 0x401F8288U
- #define **IOMUXC\_GPIO\_AD\_B1\_06\_FLEXIO1\_FLEXIO09** 0x401F8114U, 0x4U, 0, 0, 0x401F8288U
- #define **IOMUXC\_GPIO\_AD\_B1\_06\_GPIO1\_IO22** 0x401F8114U, 0x5U, 0, 0, 0x401F8288U
- #define **IOMUXC\_GPIO\_AD\_B1\_06\_LPSP11\_PCS3** 0x401F8114U, 0x6U, 0, 0, 0x401F8288U
- #define **IOMUXC\_GPIO\_AD\_B1\_06\_KPP\_COL07** 0x401F8114U, 0x7U, 0, 0, 0x401F8288U
- #define **IOMUXC\_GPIO\_AD\_B1\_07\_USDHC1\_VSELECT** 0x401F8118U, 0x0U, 0, 0, 0x401F828CU
- #define **IOMUXC\_GPIO\_AD\_B1\_07\_FLEXPWM1\_PWMB00** 0x401F8118U, 0x1U, 0x401F828CU, 0x0U, 0x401F828CU

- F8338U, 0x0U, 0x401F828CU
- #define **IOMUXC\_GPIO\_AD\_B1\_07\_LPUART2\_RTS\_B** 0x401F8118U, 0x2U, 0, 0, 0x401F828CU
- #define **IOMUXC\_GPIO\_AD\_B1\_07\_SAI1\_TX\_DATA01** 0x401F8118U, 0x3U, 0x401F8444U, 0x0U, 0x401F828CU
- #define **IOMUXC\_GPIO\_AD\_B1\_07\_FLEXIO1\_FLEXIO08** 0x401F8118U, 0x4U, 0, 0, 0x401F828CU
- #define **IOMUXC\_GPIO\_AD\_B1\_07\_GPIO1\_IO23** 0x401F8118U, 0x5U, 0, 0, 0x401F828CU
- #define **IOMUXC\_GPIO\_AD\_B1\_07\_LPSPi3\_PCS3** 0x401F8118U, 0x6U, 0, 0, 0x401F828CU
- #define **IOMUXC\_GPIO\_AD\_B1\_07\_KPP\_ROW07** 0x401F8118U, 0x7U, 0, 0, 0x401F828CU
- #define **IOMUXC\_GPIO\_AD\_B1\_08\_LPI2C2\_SCL** 0x401F811CU, 0x0U, 0x401F8384U, 0x0U, 0x401F8290U
- #define **IOMUXC\_GPIO\_AD\_B1\_08\_FLEXPWM1\_PWMA01** 0x401F811CU, 0x1U, 0x401F832CU, 0x0U, 0x401F8290U
- #define **IOMUXC\_GPIO\_AD\_B1\_08\_LPUART2\_TX** 0x401F811CU, 0x2U, 0x401F83D4U, 0x0U, 0x401F8290U
- #define **IOMUXC\_GPIO\_AD\_B1\_08\_SAI1\_TX\_DATA02** 0x401F811CU, 0x3U, 0x401F8440U, 0x0U, 0x401F8290U
- #define **IOMUXC\_GPIO\_AD\_B1\_08\_FLEXIO1\_FLEXIO07** 0x401F811CU, 0x4U, 0, 0, 0x401F8290U
- #define **IOMUXC\_GPIO\_AD\_B1\_08\_GPIO1\_IO24** 0x401F811CU, 0x5U, 0, 0, 0x401F8290U
- #define **IOMUXC\_GPIO\_AD\_B1\_08\_LPSPi3\_PCS2** 0x401F811CU, 0x6U, 0, 0, 0x401F8290U
- #define **IOMUXC\_GPIO\_AD\_B1\_08\_XBAR1\_INOUT12** 0x401F811CU, 0x7U, 0x401F84B4U, 0x1U, 0x401F8290U
- #define **IOMUXC\_GPIO\_AD\_B1\_09\_LPI2C2\_SDA** 0x401F8120U, 0x0U, 0x401F8388U, 0x0U, 0x401F8294U
- #define **IOMUXC\_GPIO\_AD\_B1\_09\_FLEXPWM1\_PWMB01** 0x401F8120U, 0x1U, 0x401F833CU, 0x0U, 0x401F8294U
- #define **IOMUXC\_GPIO\_AD\_B1\_09\_LPUART2\_RX** 0x401F8120U, 0x2U, 0x401F83D0U, 0x0U, 0x401F8294U
- #define **IOMUXC\_GPIO\_AD\_B1\_09\_SAI1\_TX\_DATA03** 0x401F8120U, 0x3U, 0x401F843CU, 0x0U, 0x401F8294U
- #define **IOMUXC\_GPIO\_AD\_B1\_09\_FLEXIO1\_FLEXIO06** 0x401F8120U, 0x4U, 0, 0, 0x401F8294U
- #define **IOMUXC\_GPIO\_AD\_B1\_09\_GPIO1\_IO25** 0x401F8120U, 0x5U, 0, 0, 0x401F8294U
- #define **IOMUXC\_GPIO\_AD\_B1\_09\_LPSPi3\_PCS1** 0x401F8120U, 0x6U, 0, 0, 0x401F8294U
- #define **IOMUXC\_GPIO\_AD\_B1\_09\_XBAR1\_INOUT13** 0x401F8120U, 0x7U, 0x401F84B8U, 0x1U, 0x401F8294U
- #define **IOMUXC\_GPIO\_AD\_B1\_10\_USB\_OTG1\_PWR** 0x401F8124U, 0x0U, 0, 0, 0x401F8298U
- #define **IOMUXC\_GPIO\_AD\_B1\_10\_FLEXPWM1\_PWMA02** 0x401F8124U, 0x1U, 0x401F8330U, 0x0U, 0x401F8298U
- #define **IOMUXC\_GPIO\_AD\_B1\_10\_LPUART4\_TX** 0x401F8124U, 0x2U, 0x401F83E8U, 0x1U, 0x401F8298U
- #define **IOMUXC\_GPIO\_AD\_B1\_10\_USDHC1\_CD\_B** 0x401F8124U, 0x3U, 0x401F8490U, 0x2U, 0x401F8298U
- #define **IOMUXC\_GPIO\_AD\_B1\_10\_FLEXIO1\_FLEXIO05** 0x401F8124U, 0x4U, 0, 0, 0x401F8298U
- #define **IOMUXC\_GPIO\_AD\_B1\_10\_GPIO1\_IO26** 0x401F8124U, 0x5U, 0, 0, 0x401F8298U
- #define **IOMUXC\_GPIO\_AD\_B1\_10\_GPT2\_CAPTURE1** 0x401F8124U, 0x6U, 0, 0, 0x401F8298U

- #define **IOMUXC\_GPIO\_AD\_B1\_11\_USB\_OTG1\_ID** 0x401F8128U, 0x0U, 0x401F82FCU, 0x1U, 0x401F829CU
- #define **IOMUXC\_GPIO\_AD\_B1\_11\_FLEXPWM1\_PWMB02** 0x401F8128U, 0x1U, 0x401F8340U, 0x0U, 0x401F829CU
- #define **IOMUXC\_GPIO\_AD\_B1\_11\_LPUART4\_RX** 0x401F8128U, 0x2U, 0x401F83E4U, 0x1U, 0x401F829CU
- #define **IOMUXC\_GPIO\_AD\_B1\_11\_USDHC1\_WP** 0x401F8128U, 0x3U, 0x401F8494U, 0x3U, 0x401F829CU
- #define **IOMUXC\_GPIO\_AD\_B1\_11\_FLEXIO1\_FLEXIO04** 0x401F8128U, 0x4U, 0, 0, 0x401F829CU
- #define **IOMUXC\_GPIO\_AD\_B1\_11\_GPIO1\_IO27** 0x401F8128U, 0x5U, 0, 0, 0x401F829CU
- #define **IOMUXC\_GPIO\_AD\_B1\_11\_GPT2\_COMPARE1** 0x401F8128U, 0x6U, 0, 0, 0x401F829CU
- #define **IOMUXC\_GPIO\_AD\_B1\_12\_USB\_OTG1\_OC** 0x401F812CU, 0x0U, 0x401F848CU, 0x1U, 0x401F82A0U
- #define **IOMUXC\_GPIO\_AD\_B1\_12\_ACMP1\_OUT** 0x401F812CU, 0x1U, 0, 0, 0x401F82A0U
- #define **IOMUXC\_GPIO\_AD\_B1\_12\_LPSPi3\_SCK** 0x401F812CU, 0x2U, 0, 0, 0x401F82A0U
- #define **IOMUXC\_GPIO\_AD\_B1\_12\_USDHC2\_CD\_B** 0x401F812CU, 0x3U, 0x401F8498U, 0x2U, 0x401F82A0U
- #define **IOMUXC\_GPIO\_AD\_B1\_12\_FLEXIO1\_FLEXIO03** 0x401F812CU, 0x4U, 0, 0, 0x401F82A0U
- #define **IOMUXC\_GPIO\_AD\_B1\_12\_GPIO1\_IO28** 0x401F812CU, 0x5U, 0, 0, 0x401F82A0U
- #define **IOMUXC\_GPIO\_AD\_B1\_12\_FLEXPWM1\_PWMA03** 0x401F812CU, 0x6U, 0x401F8334U, 0x0U, 0x401F82A0U
- #define **IOMUXC\_GPIO\_AD\_B1\_13\_LPI2C1\_HREQ** 0x401F8130U, 0x0U, 0, 0, 0x401F82A4U
- #define **IOMUXC\_GPIO\_AD\_B1\_13\_ACMP2\_OUT** 0x401F8130U, 0x1U, 0, 0, 0x401F82A4U
- #define **IOMUXC\_GPIO\_AD\_B1\_13\_LPSPi3\_PCS0** 0x401F8130U, 0x2U, 0, 0, 0x401F82A4U
- #define **IOMUXC\_GPIO\_AD\_B1\_13\_USDHC2\_WP** 0x401F8130U, 0x3U, 0x401F849CU, 0x0U, 0x401F82A4U
- #define **IOMUXC\_GPIO\_AD\_B1\_13\_FLEXIO1\_FLEXIO02** 0x401F8130U, 0x4U, 0, 0, 0x401F82A4U
- #define **IOMUXC\_GPIO\_AD\_B1\_13\_GPIO1\_IO29** 0x401F8130U, 0x5U, 0, 0, 0x401F82A4U
- #define **IOMUXC\_GPIO\_AD\_B1\_13\_FLEXPWM1\_PWMB03** 0x401F8130U, 0x6U, 0x401F8344U, 0x0U, 0x401F82A4U
- #define **IOMUXC\_GPIO\_AD\_B1\_14\_LPI2C1\_SCL** 0x401F8134U, 0x0U, 0x401F837CU, 0x1U, 0x401F82A8U
- #define **IOMUXC\_GPIO\_AD\_B1\_14\_ACMP3\_OUT** 0x401F8134U, 0x1U, 0, 0, 0x401F82A8U
- #define **IOMUXC\_GPIO\_AD\_B1\_14\_LPSPi3\_SDO** 0x401F8134U, 0x2U, 0, 0, 0x401F82A8U
- #define **IOMUXC\_GPIO\_AD\_B1\_14\_ENET\_1588\_EVENT0\_OUT** 0x401F8134U, 0x3U, 0, 0, 0x401F82A8U
- #define **IOMUXC\_GPIO\_AD\_B1\_14\_FLEXIO1\_FLEXIO01** 0x401F8134U, 0x4U, 0, 0, 0x401F82A8U
- #define **IOMUXC\_GPIO\_AD\_B1\_14\_GPIO1\_IO30** 0x401F8134U, 0x5U, 0, 0, 0x401F82A8U
- #define **IOMUXC\_GPIO\_AD\_B1\_15\_LPI2C1\_SDA** 0x401F8138U, 0x0U, 0x401F8380U, 0x1U, 0x401F82ACU
- #define **IOMUXC\_GPIO\_AD\_B1\_15\_ACMP4\_OUT** 0x401F8138U, 0x1U, 0, 0, 0x401F82ACU
- #define **IOMUXC\_GPIO\_AD\_B1\_15\_LPSPi3\_SDI** 0x401F8138U, 0x2U, 0, 0, 0x401F82ACU
- #define **IOMUXC\_GPIO\_AD\_B1\_15\_ENET\_1588\_EVENT0\_IN** 0x401F8138U, 0x3U, 0, 0, 0x401F82ACU
- #define **IOMUXC\_GPIO\_AD\_B1\_15\_FLEXIO1\_FLEXIO00** 0x401F8138U, 0x4U, 0, 0, 0x401F82ACU



```

F82ACU
• #define IOMUXC_GPIO_AD_B1_15_GPIO1_IO31 0x401F8138U, 0x5U, 0, 0, 0x401F82ACU
• #define IOMUXC_GPIO_SD_B0_00_USDHC1_DATA2 0x401F813CU, 0x0U, 0, 0, 0x401F82-
 B0U
• #define IOMUXC_GPIO_SD_B0_00_QTIMER1_TIMER0 0x401F813CU, 0x1U, 0x401F8410-
 U, 0x0U, 0x401F82B0U
• #define IOMUXC_GPIO_SD_B0_00_SAI1_MCLK 0x401F813CU, 0x2U, 0x401F8430U, 0x0U,
 0x401F82B0U
• #define IOMUXC_GPIO_SD_B0_00_SAI2_MCLK 0x401F813CU, 0x3U, 0x401F8454U, 0x0U,
 0x401F82B0U
• #define IOMUXC_GPIO_SD_B0_00_LPI2C3_SCL 0x401F813CU, 0x4U, 0x401F838CU, 0x0-
 U, 0x401F82B0U
• #define IOMUXC_GPIO_SD_B0_00_GPIO3_IO13 0x401F813CU, 0x5U, 0, 0, 0x401F82B0U
• #define IOMUXC_GPIO_SD_B0_00_FLEXSPI_A_SS1_B 0x401F813CU, 0x6U, 0, 0, 0x401-
 F82B0U
• #define IOMUXC_GPIO_SD_B0_00_XBAR1_INOUT14 0x401F813CU, 0x7U, 0x401F84A0U,
 0x0U, 0x401F82B0U
• #define IOMUXC_GPIO_SD_B0_01_USDHC1_DATA3 0x401F8140U, 0x0U, 0, 0, 0x401F82-
 B4U
• #define IOMUXC_GPIO_SD_B0_01_QTIMER1_TIMER1 0x401F8140U, 0x1U, 0x401F8414-
 U, 0x0U, 0x401F82B4U
• #define IOMUXC_GPIO_SD_B0_01_REF_24M_OUT 0x401F8140U, 0x2U, 0, 0, 0x401F82B4-
 U
• #define IOMUXC_GPIO_SD_B0_01_SAI2_RX_SYNC 0x401F8140U, 0x3U, 0x401F8460U,
 0x0U, 0x401F82B4U
• #define IOMUXC_GPIO_SD_B0_01_LPI2C3_SDA 0x401F8140U, 0x4U, 0x401F8390U, 0x0U,
 0x401F82B4U
• #define IOMUXC_GPIO_SD_B0_01_GPIO3_IO14 0x401F8140U, 0x5U, 0, 0, 0x401F82B4U
• #define IOMUXC_GPIO_SD_B0_01_FLEXSPI_B_SS1_B 0x401F8140U, 0x6U, 0, 0, 0x401-
 F82B4U
• #define IOMUXC_GPIO_SD_B0_01_XBAR1_INOUT15 0x401F8140U, 0x7U, 0x401F84A4U,
 0x0U, 0x401F82B4U
• #define IOMUXC_GPIO_SD_B0_02_USDHC1_CMD 0x401F8144U, 0x0U, 0, 0, 0x401F82B8-
 U
• #define IOMUXC_GPIO_SD_B0_02_QTIMER1_TIMER2 0x401F8144U, 0x1U, 0x401F8418-
 U, 0x0U, 0x401F82B8U
• #define IOMUXC_GPIO_SD_B0_02_LPUART7_CTS_B 0x401F8144U, 0x2U, 0, 0, 0x401F82-
 B8U
• #define IOMUXC_GPIO_SD_B0_02_SAI2_RX_BCLK 0x401F8144U, 0x3U, 0x401F8458U,
 0x0U, 0x401F82B8U
• #define IOMUXC_GPIO_SD_B0_02_LPSPI1_SCK 0x401F8144U, 0x4U, 0x401F83A0U, 0x0-
 U, 0x401F82B8U
• #define IOMUXC_GPIO_SD_B0_02_GPIO3_IO15 0x401F8144U, 0x5U, 0, 0, 0x401F82B8U
• #define IOMUXC_GPIO_SD_B0_02_ENET_MDIO 0x401F8144U, 0x6U, 0x401F8308U, 0x0-
 U, 0x401F82B8U
• #define IOMUXC_GPIO_SD_B0_02_XBAR1_INOUT16 0x401F8144U, 0x7U, 0x401F84A8U,
 0x0U, 0x401F82B8U
• #define IOMUXC_GPIO_SD_B0_03_USDHC1_CLK 0x401F8148U, 0x0U, 0, 0, 0x401F82BCU
• #define IOMUXC_GPIO_SD_B0_03_QTIMER1_TIMER3 0x401F8148U, 0x1U, 0x401F841C-
 U, 0x0U, 0x401F82BCU

```

- #define **IOMUXC\_GPIO\_SD\_B0\_03\_LPUART7\_RTS\_B** 0x401F8148U, 0x2U, 0, 0, 0x401F82BCU
- #define **IOMUXC\_GPIO\_SD\_B0\_03\_SAI2\_RX\_DATA** 0x401F8148U, 0x3U, 0x401F845CU, 0x0U, 0x401F82BCU
- #define **IOMUXC\_GPIO\_SD\_B0\_03\_LPSP1\_PCS0** 0x401F8148U, 0x4U, 0x401F839CU, 0x0U, 0x401F82BCU
- #define **IOMUXC\_GPIO\_SD\_B0\_03\_GPIO3\_IO16** 0x401F8148U, 0x5U, 0, 0, 0x401F82BCU
- #define **IOMUXC\_GPIO\_SD\_B0\_03\_ENET\_MDC** 0x401F8148U, 0x6U, 0, 0, 0x401F82BCU
- #define **IOMUXC\_GPIO\_SD\_B0\_04\_USDHC1\_DATA0** 0x401F814CU, 0x0U, 0, 0, 0x401F82C0U
- #define **IOMUXC\_GPIO\_SD\_B0\_04\_FLEXCAN2\_TX** 0x401F814CU, 0x1U, 0, 0, 0x401F82C0U
- #define **IOMUXC\_GPIO\_SD\_B0\_04\_LPUART7\_TX** 0x401F814CU, 0x2U, 0x401F8400U, 0x0U, 0x401F82C0U
- #define **IOMUXC\_GPIO\_SD\_B0\_04\_SAI2\_TX\_DATA** 0x401F814CU, 0x3U, 0, 0, 0x401F82C0U
- #define **IOMUXC\_GPIO\_SD\_B0\_04\_LPSP1\_SDO** 0x401F814CU, 0x4U, 0x401F83A8U, 0x0U, 0x401F82C0U
- #define **IOMUXC\_GPIO\_SD\_B0\_04\_GPIO3\_IO17** 0x401F814CU, 0x5U, 0, 0, 0x401F82C0U
- #define **IOMUXC\_GPIO\_SD\_B0\_04\_FLEXSPI\_B\_SS0\_B** 0x401F814CU, 0x6U, 0, 0, 0x401F82C0U
- #define **IOMUXC\_GPIO\_SD\_B0\_05\_USDHC1\_DATA1** 0x401F8150U, 0x0U, 0, 0, 0x401F82C4U
- #define **IOMUXC\_GPIO\_SD\_B0\_05\_FLEXCAN2\_RX** 0x401F8150U, 0x1U, 0x401F8324U, 0x0U, 0x401F82C4U
- #define **IOMUXC\_GPIO\_SD\_B0\_05\_LPUART7\_RX** 0x401F8150U, 0x2U, 0x401F83FCU, 0x0U, 0x401F82C4U
- #define **IOMUXC\_GPIO\_SD\_B0\_05\_SAI2\_TX\_BCLK** 0x401F8150U, 0x3U, 0x401F8464U, 0x0U, 0x401F82C4U
- #define **IOMUXC\_GPIO\_SD\_B0\_05\_LPSP1\_SDI** 0x401F8150U, 0x4U, 0x401F83A4U, 0x0U, 0x401F82C4U
- #define **IOMUXC\_GPIO\_SD\_B0\_05\_GPIO3\_IO18** 0x401F8150U, 0x5U, 0, 0, 0x401F82C4U
- #define **IOMUXC\_GPIO\_SD\_B0\_05\_FLEXSPI\_B\_DQS** 0x401F8150U, 0x6U, 0, 0, 0x401F82C4U
- #define **IOMUXC\_GPIO\_SD\_B0\_06\_USDHC1\_CD\_B** 0x401F8154U, 0x0U, 0x401F8490U, 0x0U, 0x401F82C8U
- #define **IOMUXC\_GPIO\_SD\_B0\_06\_USDHC1\_RESET\_B** 0x401F8154U, 0x1U, 0, 0, 0x401F82C8U
- #define **IOMUXC\_GPIO\_SD\_B0\_06\_REF\_32K\_OUT** 0x401F8154U, 0x2U, 0, 0, 0x401F82C8U
- #define **IOMUXC\_GPIO\_SD\_B0\_06\_SAI2\_TX\_SYNC** 0x401F8154U, 0x3U, 0x401F8468U, 0x0U, 0x401F82C8U
- #define **IOMUXC\_GPIO\_SD\_B0\_06\_WDOG1\_B** 0x401F8154U, 0x4U, 0, 0, 0x401F82C8U
- #define **IOMUXC\_GPIO\_SD\_B0\_06\_GPIO3\_IO19** 0x401F8154U, 0x5U, 0, 0, 0x401F82C8U
- #define **IOMUXC\_GPIO\_SD\_B0\_06\_XBAR1\_INOUT17** 0x401F8154U, 0x6U, 0x401F84ACU, 0x0U, 0x401F82C8U
- #define **IOMUXC\_GPIO\_SD\_B1\_00\_USDHC2\_DATA2** 0x401F8158U, 0x0U, 0, 0, 0x401F82CCU
- #define **IOMUXC\_GPIO\_SD\_B1\_00\_FLEXSPI\_B\_DATA03** 0x401F8158U, 0x1U, 0, 0, 0x401F82CCU

- #define **IOMUXC\_GPIO\_SD\_B1\_00\_LPUART6\_TX** 0x401F8158U, 0x2U, 0x401F83F8U, 0x1U, 0x401F82CCU
- #define **IOMUXC\_GPIO\_SD\_B1\_00\_XBAR1\_INOUT10** 0x401F8158U, 0x3U, 0x401F84B0U, 0x1U, 0x401F82CCU
- #define **IOMUXC\_GPIO\_SD\_B1\_00\_FLEXCAN1\_TX** 0x401F8158U, 0x4U, 0, 0, 0x401F82CCU
- #define **IOMUXC\_GPIO\_SD\_B1\_00\_GPIO3\_IO20** 0x401F8158U, 0x5U, 0, 0, 0x401F82CCU
- #define **IOMUXC\_GPIO\_SD\_B1\_01\_USDHC2\_DATA3** 0x401F815CU, 0x0U, 0, 0, 0x401F82D0U
- #define **IOMUXC\_GPIO\_SD\_B1\_01\_FLEXSPI\_B\_SCLK** 0x401F815CU, 0x1U, 0, 0, 0x401F82D0U
- #define **IOMUXC\_GPIO\_SD\_B1\_01\_LPUART6\_RX** 0x401F815CU, 0x2U, 0x401F83F4U, 0x1U, 0x401F82D0U
- #define **IOMUXC\_GPIO\_SD\_B1\_01\_FLEXSPI\_A\_SS1\_B** 0x401F815CU, 0x3U, 0, 0, 0x401F82D0U
- #define **IOMUXC\_GPIO\_SD\_B1\_01\_FLEXCAN1\_RX** 0x401F815CU, 0x4U, 0x401F8320U, 0x1U, 0x401F82D0U
- #define **IOMUXC\_GPIO\_SD\_B1\_01\_GPIO3\_IO21** 0x401F815CU, 0x5U, 0, 0, 0x401F82D0U
- #define **IOMUXC\_GPIO\_SD\_B1\_02\_USDHC2\_CMD** 0x401F8160U, 0x0U, 0, 0, 0x401F82D4U
- #define **IOMUXC\_GPIO\_SD\_B1\_02\_FLEXSPI\_B\_DATA00** 0x401F8160U, 0x1U, 0, 0, 0x401F82D4U
- #define **IOMUXC\_GPIO\_SD\_B1\_02\_LPUART8\_TX** 0x401F8160U, 0x2U, 0x401F8408U, 0x0U, 0x401F82D4U
- #define **IOMUXC\_GPIO\_SD\_B1\_02\_LPI2C4\_SCL** 0x401F8160U, 0x3U, 0x401F8394U, 0x1U, 0x401F82D4U
- #define **IOMUXC\_GPIO\_SD\_B1\_02\_ENET\_1588\_EVENT1\_OUT** 0x401F8160U, 0x4U, 0, 0, 0x401F82D4U
- #define **IOMUXC\_GPIO\_SD\_B1\_02\_GPIO3\_IO22** 0x401F8160U, 0x5U, 0, 0, 0x401F82D4U
- #define **IOMUXC\_GPIO\_SD\_B1\_02\_CCM\_CLKO1** 0x401F8160U, 0x6U, 0, 0, 0x401F82D4U
- #define **IOMUXC\_GPIO\_SD\_B1\_03\_USDHC2\_CLK** 0x401F8164U, 0x0U, 0, 0, 0x401F82D8U
- #define **IOMUXC\_GPIO\_SD\_B1\_03\_FLEXSPI\_B\_DATA02** 0x401F8164U, 0x1U, 0, 0, 0x401F82D8U
- #define **IOMUXC\_GPIO\_SD\_B1\_03\_LPUART8\_RX** 0x401F8164U, 0x2U, 0x401F8404U, 0x0U, 0x401F82D8U
- #define **IOMUXC\_GPIO\_SD\_B1\_03\_LPI2C4\_SDA** 0x401F8164U, 0x3U, 0x401F8398U, 0x1U, 0x401F82D8U
- #define **IOMUXC\_GPIO\_SD\_B1\_03\_ENET\_1588\_EVENT1\_IN** 0x401F8164U, 0x4U, 0, 0, 0x401F82D8U
- #define **IOMUXC\_GPIO\_SD\_B1\_03\_GPIO3\_IO23** 0x401F8164U, 0x5U, 0, 0, 0x401F82D8U
- #define **IOMUXC\_GPIO\_SD\_B1\_03\_CCM\_CLKO2** 0x401F8164U, 0x6U, 0, 0, 0x401F82D8U
- #define **IOMUXC\_GPIO\_SD\_B1\_04\_USDHC2\_DATA0** 0x401F8168U, 0x0U, 0, 0, 0x401F82DCU
- #define **IOMUXC\_GPIO\_SD\_B1\_04\_FLEXSPI\_B\_DATA01** 0x401F8168U, 0x1U, 0, 0, 0x401F82DCU
- #define **IOMUXC\_GPIO\_SD\_B1\_04\_ENET\_TX\_CLK** 0x401F8168U, 0x2U, 0x401F831CU, 0x0U, 0x401F82DCU
- #define **IOMUXC\_GPIO\_SD\_B1\_04\_ENET\_REF\_CLK1** 0x401F8168U, 0x3U, 0x401F8304U, 0x0U, 0x401F82DCU
- #define **IOMUXC\_GPIO\_SD\_B1\_04\_EWM\_OUT\_B** 0x401F8168U, 0x4U, 0, 0, 0x401F82DCU
- #define **IOMUXC\_GPIO\_SD\_B1\_04\_GPIO3\_IO24** 0x401F8168U, 0x5U, 0, 0, 0x401F82DCU

- #define **IOMUXC\_GPIO\_SD\_B1\_04\_CCM\_WAIT** 0x401F8168U, 0x6U, 0, 0, 0x401F82DCU
- #define **IOMUXC\_GPIO\_SD\_B1\_05\_USDHC2\_DATA1** 0x401F816CU, 0x0U, 0, 0, 0x401F82E0U
- #define **IOMUXC\_GPIO\_SD\_B1\_05\_FLEXSPI\_A\_DQS** 0x401F816CU, 0x1U, 0, 0, 0x401F82E0U
- #define **IOMUXC\_GPIO\_SD\_B1\_05\_ENET\_RDATA01** 0x401F816CU, 0x2U, 0x401F8310U, 0x0U, 0x401F82E0U
- #define **IOMUXC\_GPIO\_SD\_B1\_05\_SAI3\_MCLK** 0x401F816CU, 0x3U, 0x401F846CU, 0x0U, 0x401F82E0U
- #define **IOMUXC\_GPIO\_SD\_B1\_05\_FLEXSPI\_B\_SS0\_B** 0x401F816CU, 0x4U, 0, 0, 0x401F82E0U
- #define **IOMUXC\_GPIO\_SD\_B1\_05\_GPIO3\_IO25** 0x401F816CU, 0x5U, 0, 0, 0x401F82E0U
- #define **IOMUXC\_GPIO\_SD\_B1\_05\_CCM\_PMIC\_RDY** 0x401F816CU, 0x6U, 0x401F8300U, 0x1U, 0x401F82E0U
- #define **IOMUXC\_GPIO\_SD\_B1\_06\_USDHC2\_CD\_B** 0x401F8170U, 0x0U, 0x401F8498U, 0x0U, 0x401F82E4U
- #define **IOMUXC\_GPIO\_SD\_B1\_06\_FLEXSPI\_A\_DATA03** 0x401F8170U, 0x1U, 0x401F8374U, 0x0U, 0x401F82E4U
- #define **IOMUXC\_GPIO\_SD\_B1\_06\_ENET\_RDATA00** 0x401F8170U, 0x2U, 0x401F830CU, 0x0U, 0x401F82E4U
- #define **IOMUXC\_GPIO\_SD\_B1\_06\_SAI3\_TX\_BCLK** 0x401F8170U, 0x3U, 0x401F847CU, 0x0U, 0x401F82E4U
- #define **IOMUXC\_GPIO\_SD\_B1\_06\_LPSP12\_PCS0** 0x401F8170U, 0x4U, 0x401F83ACU, 0x2U, 0x401F82E4U
- #define **IOMUXC\_GPIO\_SD\_B1\_06\_GPIO3\_IO26** 0x401F8170U, 0x5U, 0, 0, 0x401F82E4U
- #define **IOMUXC\_GPIO\_SD\_B1\_06\_CCM\_STOP** 0x401F8170U, 0x6U, 0, 0, 0x401F82E4U
- #define **IOMUXC\_GPIO\_SD\_B1\_07\_USDHC2\_RESET\_B** 0x401F8174U, 0x0U, 0, 0, 0x401F82E8U
- #define **IOMUXC\_GPIO\_SD\_B1\_07\_FLEXSPI\_A\_SCLK** 0x401F8174U, 0x1U, 0x401F8378U, 0x0U, 0x401F82E8U
- #define **IOMUXC\_GPIO\_SD\_B1\_07\_ENET\_RX\_EN** 0x401F8174U, 0x2U, 0x401F8314U, 0x0U, 0x401F82E8U
- #define **IOMUXC\_GPIO\_SD\_B1\_07\_SAI3\_TX\_SYNC** 0x401F8174U, 0x3U, 0x401F8480U, 0x0U, 0x401F82E8U
- #define **IOMUXC\_GPIO\_SD\_B1\_07\_LPSP12\_SCK** 0x401F8174U, 0x4U, 0, 0, 0x401F82E8U
- #define **IOMUXC\_GPIO\_SD\_B1\_07\_GPIO3\_IO27** 0x401F8174U, 0x5U, 0, 0, 0x401F82E8U
- #define **IOMUXC\_GPIO\_SD\_B1\_08\_USDHC2\_DATA4** 0x401F8178U, 0x0U, 0, 0, 0x401F82ECU
- #define **IOMUXC\_GPIO\_SD\_B1\_08\_FLEXSPI\_A\_DATA00** 0x401F8178U, 0x1U, 0x401F8368U, 0x0U, 0x401F82ECU
- #define **IOMUXC\_GPIO\_SD\_B1\_08\_ENET\_RX\_ER** 0x401F8178U, 0x2U, 0x401F8318U, 0x0U, 0x401F82ECU
- #define **IOMUXC\_GPIO\_SD\_B1\_08\_SAI3\_TX\_DATA** 0x401F8178U, 0x3U, 0, 0, 0x401F82ECU
- #define **IOMUXC\_GPIO\_SD\_B1\_08\_LPSP12\_SDO** 0x401F8178U, 0x4U, 0x401F83B8U, 0x2U, 0x401F82ECU
- #define **IOMUXC\_GPIO\_SD\_B1\_08\_GPIO3\_IO28** 0x401F8178U, 0x5U, 0, 0, 0x401F82ECU
- #define **IOMUXC\_GPIO\_SD\_B1\_09\_USDHC2\_DATA5** 0x401F817CU, 0x0U, 0, 0, 0x401F82F0U
- #define **IOMUXC\_GPIO\_SD\_B1\_09\_FLEXSPI\_A\_DATA02** 0x401F817CU, 0x1U, 0x401F8374U, 0x0U, 0x401F82E4U

- F8370U, 0x0U, 0x401F82F0U
- #define **IOMUXC\_GPIO\_SD\_B1\_09\_ENET\_TX\_EN** 0x401F817CU, 0x2U, 0, 0, 0x401F82F0U
- #define **IOMUXC\_GPIO\_SD\_B1\_09\_SAI3\_RX\_BCLK** 0x401F817CU, 0x3U, 0x401F8470U, 0x0U, 0x401F82F0U
- #define **IOMUXC\_GPIO\_SD\_B1\_09\_LPSPi2\_SDI** 0x401F817CU, 0x4U, 0x401F83B4U, 0x2U, 0x401F82F0U
- #define **IOMUXC\_GPIO\_SD\_B1\_09\_GPIO3\_IO29** 0x401F817CU, 0x5U, 0, 0, 0x401F82F0U
- #define **IOMUXC\_GPIO\_SD\_B1\_10\_USDHC2\_DATA6** 0x401F8180U, 0x0U, 0, 0, 0x401F82F4U
- #define **IOMUXC\_GPIO\_SD\_B1\_10\_FLEXSPI\_A\_DATA01** 0x401F8180U, 0x1U, 0x401F836CU, 0x0U, 0x401F82F4U
- #define **IOMUXC\_GPIO\_SD\_B1\_10\_ENET\_TDATA00** 0x401F8180U, 0x2U, 0, 0, 0x401F82F4U
- #define **IOMUXC\_GPIO\_SD\_B1\_10\_SAI3\_RX\_SYNC** 0x401F8180U, 0x3U, 0x401F8478U, 0x0U, 0x401F82F4U
- #define **IOMUXC\_GPIO\_SD\_B1\_10\_LPSPi2\_PCS2** 0x401F8180U, 0x4U, 0, 0, 0x401F82F4U
- #define **IOMUXC\_GPIO\_SD\_B1\_10\_GPIO3\_IO30** 0x401F8180U, 0x5U, 0, 0, 0x401F82F4U
- #define **IOMUXC\_GPIO\_SD\_B1\_11\_USDHC2\_DATA7** 0x401F8184U, 0x0U, 0, 0, 0x401F82F8U
- #define **IOMUXC\_GPIO\_SD\_B1\_11\_FLEXSPI\_A\_SS0\_B** 0x401F8184U, 0x1U, 0, 0, 0x401F82F8U
- #define **IOMUXC\_GPIO\_SD\_B1\_11\_ENET\_TDATA01** 0x401F8184U, 0x2U, 0, 0, 0x401F82F8U
- #define **IOMUXC\_GPIO\_SD\_B1\_11\_SAI3\_RX\_DATA** 0x401F8184U, 0x3U, 0x401F8474U, 0x0U, 0x401F82F8U
- #define **IOMUXC\_GPIO\_SD\_B1\_11\_LPSPi2\_PCS3** 0x401F8184U, 0x4U, 0, 0, 0x401F82F8U
- #define **IOMUXC\_GPIO\_SD\_B1\_11\_GPIO3\_IO31** 0x401F8184U, 0x5U, 0, 0, 0x401F82F8U
- #define **IOMUXC\_SNVS\_WAKEUP\_GPIO5\_IO00** 0x400A8000U, 0x5U, 0, 0, 0x400A8018U
- #define **IOMUXC\_SNVS\_WAKEUP\_NMI\_GLUE\_NMI** 0x400A8000U, 0x7U, 0x401F840CU, 0x1U, 0x400A8018U
- #define **IOMUXC\_SNVS\_PMIC\_ON\_REQ\_SNVS\_LP\_PMIC\_ON\_REQ** 0x400A8004U, 0x0U, 0, 0, 0x400A801CU
- #define **IOMUXC\_SNVS\_PMIC\_ON\_REQ\_GPIO5\_IO01** 0x400A8004U, 0x5U, 0, 0, 0x400A801CU
- #define **IOMUXC\_SNVS\_PMIC\_STBY\_REQ\_CCM\_PMIC\_VSTBY\_REQ** 0x400A8008U, 0x0U, 0, 0, 0x400A8020U
- #define **IOMUXC\_SNVS\_PMIC\_STBY\_REQ\_GPIO5\_IO02** 0x400A8008U, 0x5U, 0, 0, 0x400A8020U
- #define **IOMUXC\_SNVS\_TEST\_MODE** 0, 0, 0, 0, 0x400A800CU
- #define **IOMUXC\_SNVS\_POR\_B** 0, 0, 0, 0, 0x400A8010U
- #define **IOMUXC\_SNVS\_ONOFF** 0, 0, 0, 0, 0x400A8014U

## Configuration

- static void [IOMUXC\\_SetPinMux](#) (uint32\_t muxRegister, uint32\_t muxMode, uint32\_t inputRegister, uint32\_t inputDaisy, uint32\_t configRegister, uint32\_t inputOnfield)  
*Sets the IOMUXC pin mux mode.*
- static void [IOMUXC\\_SetPinConfig](#) (uint32\_t muxRegister, uint32\_t muxMode, uint32\_t inputRegister, uint32\_t inputDaisy, uint32\_t configRegister, uint32\_t configValue)  
*Sets the IOMUXC pin configuration.*
- static void [IOMUXC\\_EnableMode](#) (IOMUXC\_GPR\_Type \*base, uint32\_t mode, bool enable)

- Sets IOMUXC general configuration for some mode.*

• static void [IOMUXC\\_SetSaiMClkClockSource](#) (IOMUXC\_GPR\_Type \*base, iomuxc\_gpr\_saimclk\_t mclk, uint8\_t clkSrc)

*Sets IOMUXC general configuration for SAI MCLK selection.*
- static void [IOMUXC\\_MQSEnterSoftwareReset](#) (IOMUXC\_GPR\_Type \*base, bool enable)

*Enters or exit MQS software reset.*
- static void [IOMUXC\\_MQSEnable](#) (IOMUXC\_GPR\_Type \*base, bool enable)

*Enables or disables MQS.*
- static void [IOMUXC\\_MQSConfig](#) (IOMUXC\_GPR\_Type \*base, iomuxc\_mqs\_pwm\_oversample\_rate\_t rate, uint8\_t divider)

*Configure MQS PWM oversampling rate compared with mclk and divider ratio control for mclk from hmclk.*

## 6.2 Macro Definition Documentation

### 6.2.1 #define FSL\_IOMUXC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))

## 6.3 Function Documentation

### 6.3.1 static void IOMUXC\_SetPinMux ( uint32\_t muxRegister, uint32\_t muxMode, uint32\_t inputRegister, uint32\_t inputDaisy, uint32\_t configRegister, uint32\_t inputOnfield ) [inline], [static]

Note

The first five parameters can be filled with the pin function ID macros.

This is an example to set the PTA6 as the lpuart0\_tx:

```
* IOMUXC_SetPinMux (IOMUXC_PTA6_LPUART0_TX, 0);
*
```

This is an example to set the PTA0 as GPIOA0:

```
* IOMUXC_SetPinMux (IOMUXC_PTA0_GPIOA0, 0);
*
```

Parameters

|                    |                       |
|--------------------|-----------------------|
| <i>muxRegister</i> | The pin mux register. |
| <i>muxMode</i>     | The pin mux mode.     |

|                       |                            |
|-----------------------|----------------------------|
| <i>inputRegister</i>  | The select input register. |
| <i>inputDaisy</i>     | The input daisy.           |
| <i>configRegister</i> | The config register.       |
| <i>inputOnfield</i>   | Software input on field.   |

**6.3.2 static void IOMUXC\_SetPinConfig ( uint32\_t muxRegister, uint32\_t muxMode, uint32\_t inputRegister, uint32\_t inputDaisy, uint32\_t configRegister, uint32\_t configValue ) [inline], [static]**

Note

The previous five parameters can be filled with the pin function ID macros.

This is an example to set pin configuration for IOMUXC\_PTA3\_LPI2C0\_SCL5:

```
* IOMUXC_SetPinConfig(IOMUXC_PTA3_LPI2C0_SCL5, IOMUXC_SW_PAD_CTL_PAD_PUS_MASK |
 IOMUXC_SW_PAD_CTL_PAD_PUS(2U))
*
```

Parameters

|                       |                            |
|-----------------------|----------------------------|
| <i>muxRegister</i>    | The pin mux register.      |
| <i>muxMode</i>        | The pin mux mode.          |
| <i>inputRegister</i>  | The select input register. |
| <i>inputDaisy</i>     | The input daisy.           |
| <i>configRegister</i> | The config register.       |
| <i>configValue</i>    | The pin config value.      |

**6.3.3 static void IOMUXC\_EnableMode ( IOMUXC\_GPR\_Type \* base, uint32\_t mode, bool enable ) [inline], [static]**

Parameters

|               |                                                                       |
|---------------|-----------------------------------------------------------------------|
| <i>base</i>   | The IOMUXC GPR base address.                                          |
| <i>mode</i>   | The mode for setting. the mode is the logical OR of "iomuxc_gpr_mode" |
| <i>enable</i> | True enable false disable.                                            |

**6.3.4 static void IOMUXC\_SetSaiMClkClockSource ( IOMUXC\_GPR\_Type \* *base*, iomuxc\_gpr\_saimclk\_t *mclk*, uint8\_t *clkSrc* ) [inline], [static]**

Parameters

|               |                                                                                      |
|---------------|--------------------------------------------------------------------------------------|
| <i>base</i>   | The IOMUXC GPR base address.                                                         |
| <i>mclk</i>   | The SAI MCLK.                                                                        |
| <i>clkSrc</i> | The clock source. Take refer to register setting details for the clock source in RM. |

**6.3.5 static void IOMUXC\_MQSEnterSoftwareReset ( IOMUXC\_GPR\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | The IOMUXC GPR base address.      |
| <i>enable</i> | Enter or exit MQS software reset. |

**6.3.6 static void IOMUXC\_MQSEnable ( IOMUXC\_GPR\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | The IOMUXC GPR base address. |
| <i>enable</i> | Enable or disable the MQS.   |

**6.3.7 static void IOMUXC\_MQSConfig ( IOMUXC\_GPR\_Type \* *base*, iomuxc\_mqs\_pwm\_oversample\_rate\_t *rate*, uint8\_t *divider* ) [inline], [static]**



Parameters

|                |                                                                                                 |
|----------------|-------------------------------------------------------------------------------------------------|
| <i>base</i>    | The IOMUXC GPR base address.                                                                    |
| <i>rate</i>    | The MQS PWM oversampling rate, refer to "iomuxc_mqs_pwm_oversample_rate_t".                     |
| <i>divider</i> | The divider ratio control for mclk from hmclk. $mclk\ freq = 1 / (divider + 1) * hmclk\ freq$ . |

## Chapter 7

# ADC: 12-bit Analog to Digital Converter Driver

### 7.1 Overview

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

### 7.2 Typical use case

#### 7.2.1 Polling Configuration

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

#### 7.2.2 Polling Configuration

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

### Data Structures

- struct `adc_config_t`  
*Converter configuration. [More...](#)*
- struct `adc_offset_config_t`  
*Converter Offset configuration. [More...](#)*
- struct `adc_hardware_compare_config_t`  
*ADC hardware compare configuration. [More...](#)*
- struct `adc_channel_config_t`  
*ADC channel conversion configuration. [More...](#)*

### Macros

- `#define FSL_ADC_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))`  
*ADC driver version.*

### Enumerations

- enum `adc_status_flags_t` {  
    `kADC_ConversionActiveFlag` = `ADC_GS_ADACT_MASK`,  
    `kADC_CalibrationFailedFlag` = `ADC_GS_CALF_MASK`,  
    `kADC_AsynchronousWakeupInterruptFlag` }  
*Converter's status flags.*

- enum `adc_reference_voltage_source_t` { `kADC_ReferenceVoltageSourceAlt0 = 0U` }  
*Reference voltage source.*
- enum `adc_sample_period_mode_t` {  
`kADC_SamplePeriod2or12Clocks = 0U,`  
`kADC_SamplePeriod4or16Clocks = 1U,`  
`kADC_SamplePeriod6or20Clocks = 2U,`  
`kADC_SamplePeriod8or24Clocks = 3U,`  
`kADC_SamplePeriodLong12Clcoks = kADC_SamplePeriod2or12Clocks,`  
`kADC_SamplePeriodLong16Clcoks = kADC_SamplePeriod4or16Clocks,`  
`kADC_SamplePeriodLong20Clcoks = kADC_SamplePeriod6or20Clocks,`  
`kADC_SamplePeriodLong24Clcoks = kADC_SamplePeriod8or24Clocks,`  
`kADC_SamplePeriodShort2Clocks = kADC_SamplePeriod2or12Clocks,`  
`kADC_SamplePeriodShort4Clocks = kADC_SamplePeriod4or16Clocks,`  
`kADC_SamplePeriodShort6Clocks = kADC_SamplePeriod6or20Clocks,`  
`kADC_SamplePeriodShort8Clocks = kADC_SamplePeriod8or24Clocks` }  
*Sample time duration.*
- enum `adc_clock_source_t` {  
`kADC_ClockSourceIPG = 0U,`  
`kADC_ClockSourceIPGDiv2 = 1U,`  
`kADC_ClockSourceALT = 2U,`  
`kADC_ClockSourceAD = 3U` }  
*Clock source.*
- enum `adc_clock_driver_t` {  
`kADC_ClockDriver1 = 0U,`  
`kADC_ClockDriver2 = 1U,`  
`kADC_ClockDriver4 = 2U,`  
`kADC_ClockDriver8 = 3U` }  
*Clock divider for the converter.*
- enum `adc_resolution_t` {  
`kADC_Resolution8Bit = 0U,`  
`kADC_Resolution10Bit = 1U,`  
`kADC_Resolution12Bit = 2U` }  
*Converter's resolution.*
- enum `adc_hardware_compare_mode_t` {  
`kADC_HardwareCompareMode0 = 0U,`  
`kADC_HardwareCompareMode1 = 1U,`  
`kADC_HardwareCompareMode2 = 2U,`  
`kADC_HardwareCompareMode3 = 3U` }  
*Converter hardware compare mode.*
- enum `adc_hardware_average_mode_t` {  
`kADC_HardwareAverageCount4 = 0U,`  
`kADC_HardwareAverageCount8 = 1U,`  
`kADC_HardwareAverageCount16 = 2U,`  
`kADC_HardwareAverageCount32 = 3U,`  
`kADC_HardwareAverageDiasable = 4U` }  
*Converter hardware average mode.*

## Initialization

- void [ADC\\_Init](#) (ADC\_Type \*base, const [adc\\_config\\_t](#) \*config)  
*Initialize the ADC module.*
- void [ADC\\_Deinit](#) (ADC\_Type \*base)  
*De-initializes the ADC module.*
- void [ADC\\_GetDefaultConfig](#) ([adc\\_config\\_t](#) \*config)  
*Gets an available pre-defined settings for the converter's configuration.*
- void [ADC\\_SetChannelConfig](#) (ADC\_Type \*base, uint32\_t channelGroup, const [adc\\_channel\\_config\\_t](#) \*config)  
*Configures the conversion channel.*
- static uint32\_t [ADC\\_GetChannelConversionValue](#) (ADC\_Type \*base, uint32\_t channelGroup)  
*Gets the conversion value.*
- static uint32\_t [ADC\\_GetChannelStatusFlags](#) (ADC\_Type \*base, uint32\_t channelGroup)  
*Gets the status flags of channel.*
- [status\\_t](#) [ADC\\_DoAutoCalibration](#) (ADC\_Type \*base)  
*Automates the hardware calibration.*
- void [ADC\\_SetOffsetConfig](#) (ADC\_Type \*base, const [adc\\_offset\\_config\\_t](#) \*config)  
*Set user defined offset.*
- static void [ADC\\_EnableDMA](#) (ADC\_Type \*base, bool enable)  
*Enables generating the DMA trigger when the conversion is complete.*
- static void [ADC\\_EnableHardwareTrigger](#) (ADC\_Type \*base, bool enable)  
*Enables the hardware trigger mode.*
- void [ADC\\_SetHardwareCompareConfig](#) (ADC\_Type \*base, const [adc\\_hardware\\_compare\\_config\\_t](#) \*config)  
*Configures the hardware compare mode.*
- void [ADC\\_SetHardwareAverageConfig](#) (ADC\_Type \*base, [adc\\_hardware\\_average\\_mode\\_t](#) mode)  
*Configures the hardware average mode.*
- static uint32\_t [ADC\\_GetStatusFlags](#) (ADC\_Type \*base)  
*Gets the converter's status flags.*
- void [ADC\\_ClearStatusFlags](#) (ADC\_Type \*base, uint32\_t mask)  
*Clears the converter's status falgs.*

## 7.3 Data Structure Documentation

### 7.3.1 struct [adc\\_config\\_t](#)

#### Data Fields

- bool [enableOverWrite](#)  
*Enable the overwriting.*
- bool [enableContinuousConversion](#)  
*Enable the continuous conversion mode.*
- bool [enableHighSpeed](#)  
*Enable the high-speed mode.*
- bool [enableLowPower](#)  
*Enable the low power mode.*
- bool [enableLongSample](#)  
*Enable the long sample mode.*
- bool [enableAsynchronousClockOutput](#)

- *Enable the asynchronous clock output.*
- [adc\\_reference\\_voltage\\_source\\_t](#) `referenceVoltageSource`  
*Select the reference voltage source.*
- [adc\\_sample\\_period\\_mode\\_t](#) `samplePeriodMode`  
*Select the sample period in long sample mode or short mode.*
- [adc\\_clock\\_source\\_t](#) `clockSource`  
*Select the input clock source to generate the internal clock ADCK.*
- [adc\\_clock\\_driver\\_t](#) `clockDriver`  
*Select the divide ratio used by the ADC to generate the internal clock ADCK.*
- [adc\\_resolution\\_t](#) `resolution`  
*Select the ADC resolution mode.*

### Field Documentation

- (1) `bool adc_config_t::enableOverWrite`
- (2) `bool adc_config_t::enableContinuousConversion`
- (3) `bool adc_config_t::enableHighSpeed`
- (4) `bool adc_config_t::enableLowPower`
- (5) `bool adc_config_t::enableLongSample`
- (6) `bool adc_config_t::enableAsynchronousClockOutput`
- (7) `adc_reference_voltage_source_t adc_config_t::referenceVoltageSource`
- (8) `adc_sample_period_mode_t adc_config_t::samplePeriodMode`
- (9) `adc_clock_source_t adc_config_t::clockSource`
- (10) `adc_clock_driver_t adc_config_t::clockDriver`
- (11) `adc_resolution_t adc_config_t::resolution`

### 7.3.2 struct `adc_offset_config_t`

#### Data Fields

- `bool enableSigned`  
*if false, The offset value is added with the raw result.*
- `uint32_t offsetValue`  
*User configurable offset value(0-4095).*

#### Field Documentation

- (1) `bool adc_offset_config_t::enableSigned`

if true, The offset value is subtracted from the raw converted value.

(2) `uint32_t adc_offest_config_t::offsetValue`

### 7.3.3 struct `adc_hardware_compare_config_t`

In `kADC_HardwareCompareMode0`, compare true if the result is less than the value1. In `kADC_HardwareCompareMode1`, compare true if the result is greater than or equal to value1. In `kADC_HardwareCompareMode2`, Value1 <= Value2, compare true if the result is less than value1 Or the result is Greater than value2. Value1 > Value2, compare true if the result is less than value1 And the result is Greater than value2. In `kADC_HardwareCompareMode3`, Value1 <= Value2, compare true if the result is greater than or equal to value1 And the result is less than or equal to value2. Value1 > Value2, compare true if the result is greater than or equal to value1 Or the result is less than or equal to value2.

#### Data Fields

- `adc_hardware_compare_mode_t hardwareCompareMode`  
*Select the hardware compare mode.*
- `uint16_t value1`  
*Setting value1(0-4095) for hardware compare mode.*
- `uint16_t value2`  
*Setting value2(0-4095) for hardware compare mode.*

#### Field Documentation

(1) `adc_hardware_compare_mode_t adc_hardware_compare_config_t::hardwareCompareMode`

See "`adc_hardware_compare_mode_t`".

(2) `uint16_t adc_hardware_compare_config_t::value1`

(3) `uint16_t adc_hardware_compare_config_t::value2`

### 7.3.4 struct `adc_channel_config_t`

#### Data Fields

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

#### Field Documentation

(1) `uint32_t adc_channel_config_t::channelNumber`

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

(2) `bool adc_channel_config_t::enableInterruptOnConversionCompleted`

## 7.4 Macro Definition Documentation

### 7.4.1 `#define FSL_ADC_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))`

Version 2.0.4.

## 7.5 Enumeration Type Documentation

### 7.5.1 `enum adc_status_flags_t`

Enumerator

*kADC\_ConversionActiveFlag* Conversion is active,not support wlc.

*kADC\_CalibrationFailedFlag* Calibration is failed,support wlc.

*kADC\_AsynchronousWakeupInterruptFlag* Asynchronous wakeup interrupt occurred, support wlc.

### 7.5.2 `enum adc_reference_voltage_source_t`

Enumerator

*kADC\_ReferenceVoltageSourceAlt0* For external pins pair of VrefH and VrefL.

### 7.5.3 `enum adc_sample_period_mode_t`

Enumerator

*kADC\_SamplePeriod2or12Clocks* Long sample 12 clocks or short sample 2 clocks.

*kADC\_SamplePeriod4or16Clocks* Long sample 16 clocks or short sample 4 clocks.

*kADC\_SamplePeriod6or20Clocks* Long sample 20 clocks or short sample 6 clocks.

*kADC\_SamplePeriod8or24Clocks* Long sample 24 clocks or short sample 8 clocks.

*kADC\_SamplePeriodLong12Clcoks* Long sample 12 clocks.

*kADC\_SamplePeriodLong16Clcoks* Long sample 16 clocks.

*kADC\_SamplePeriodLong20Clcoks* Long sample 20 clocks.

*kADC\_SamplePeriodLong24Clcoks* Long sample 24 clocks.

*kADC\_SamplePeriodShort2Clocks* Short sample 2 clocks.

*kADC\_SamplePeriodShort4Clocks* Short sample 4 clocks.

*kADC\_SamplePeriodShort6Clocks* Short sample 6 clocks.

*kADC\_SamplePeriodShort8Clocks* Short sample 8 clocks.

### 7.5.4 enum adc\_clock\_source\_t

Enumerator

- kADC\_ClockSourceIPG* Select IPG clock to generate ADCK.
- kADC\_ClockSourceIPGDiv2* Select IPG clock divided by 2 to generate ADCK.
- kADC\_ClockSourceALT* Select alternate clock to generate ADCK.
- kADC\_ClockSourceAD* Select Asynchronous clock to generate ADCK.

### 7.5.5 enum adc\_clock\_driver\_t

Enumerator

- kADC\_ClockDriver1* For divider 1 from the input clock to the module.
- kADC\_ClockDriver2* For divider 2 from the input clock to the module.
- kADC\_ClockDriver4* For divider 4 from the input clock to the module.
- kADC\_ClockDriver8* For divider 8 from the input clock to the module.

### 7.5.6 enum adc\_resolution\_t

Enumerator

- kADC\_Resolution8Bit* Single End 8-bit resolution.
- kADC\_Resolution10Bit* Single End 10-bit resolution.
- kADC\_Resolution12Bit* Single End 12-bit resolution.

### 7.5.7 enum adc\_hardware\_compare\_mode\_t

Enumerator

- kADC\_HardwareCompareMode0* Compare true if the result is less than the value1.
- kADC\_HardwareCompareMode1* Compare true if the result is greater than or equal to value1.
- kADC\_HardwareCompareMode2* Value1 <= Value2, compare true if the result is less than value1 Or the result is Greater than value2. Value1 > Value2, compare true if the result is less than value1 And the result is greater than value2
- kADC\_HardwareCompareMode3* Value1 <= Value2, compare true if the result is greater than or equal to value1 And the result is less than or equal to value2. Value1 > Value2, compare true if the result is greater than or equal to value1 Or the result is less than or equal to value2.



## 7.5.8 enum adc\_hardware\_average\_mode\_t

Enumerator

- kADC\_HardwareAverageCount4*** For hardware average with 4 samples.
- kADC\_HardwareAverageCount8*** For hardware average with 8 samples.
- kADC\_HardwareAverageCount16*** For hardware average with 16 samples.
- kADC\_HardwareAverageCount32*** For hardware average with 32 samples.
- kADC\_HardwareAverageDiasable*** Disable the hardware average function.

## 7.6 Function Documentation

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

Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | ADC peripheral base address.         |
| <i>config</i> | Pointer to "adc_config_t" structure. |

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

Parameters

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

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

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

```
* config->enableAsynchronousClockOutput = true;
* config->enableOverWrite = false;
* config->enableContinuousConversion = false;
* config->enableHighSpeed = false;
* config->enableLowPower = false;
* config->enableLongSample = false;
* config->referenceVoltageSource = kADC_ReferenceVoltageSourceAlt0;
* config->samplePeriodMode = kADC_SamplePeriod2or12Clocks;
* config->clockSource = kADC_ClockSourceAD;
* config->clockDriver = kADC_ClockDriver1;
* config->resolution = kADC_Resolution12Bit;
*
```

## Parameters

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

#### 7.6.4 void ADC\_SetChannelConfig ( ADC\_Type \* *base*, uint32\_t *channelGroup*, const adc\_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 groups 1 and greater indicate potentially multiple channel group registers for use only in hardware trigger mode. See the chip configuration information in the appropriate MCU reference manual about the number of SC1n registers (channel groups) specific to this device. None of the channel groups 1 or greater are used for software trigger operation. Therefore, writing to these channel groups does not initiate a new conversion. Updating the channel group 0 while a different channel group is actively controlling a conversion is allowed and vice versa. Writing any of the channel group registers while that specific channel group is actively controlling a conversion aborts the current conversion.

## Parameters

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

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

## Parameters

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

Returns

Conversion value.

### 7.6.6 `static uint32_t ADC_GetChannelStatusFlags ( ADC_Type * base, uint32_t channelGroup ) [inline], [static]`

A conversion is completed when the result of the conversion is transferred into the data result registers. (provided the compare function & hardware averaging is disabled), this is indicated by the setting of COCON. If hardware averaging is enabled, COCON sets only, if the last of the selected number of conversions is complete. If the compare function is enabled, COCON sets and conversion result data is transferred only if the compare condition is true. If both hardware averaging and compare functions are enabled, then COCON sets only if the last of the selected number of conversions is complete and the compare condition is true.

Parameters

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

Returns

Status flags of channel. return 0 means COCO flag is 0, return 1 means COCOflag is 1.

### 7.6.7 `status_t ADC_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 software trigger should be used during calibration.

Parameters

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

Returns

Execution status.

Return values

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

**7.6.8 void ADC\_SetOffsetConfig ( ADC\_Type \* *base*, const adc\_offset\_config\_t \* *config* )**

Parameters

|               |                                             |
|---------------|---------------------------------------------|
| <i>base</i>   | ADC peripheral base address.                |
| <i>config</i> | Pointer to "adc_offset_config_t" structure. |

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

Parameters

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

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

Parameters

|               |                                                                                                |
|---------------|------------------------------------------------------------------------------------------------|
| <i>base</i>   | ADC peripheral base address.                                                                   |
| <i>enable</i> | Switcher of the trigger mode. "true" means hardware trigger mode, "false" means software mode. |

**7.6.11 void ADC\_SetHardwareCompareConfig ( ADC\_Type \* *base*, const adc\_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 "adc\_hardware\_compare\_mode\_t" or the appropriate reference manual for more information.

## Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | ADC peripheral base address.                          |
| <i>config</i> | Pointer to "adc_hardware_compare_config_t" structure. |

### 7.6.12 void ADC\_SetHardwareAverageConfig ( ADC\_Type \* *base*, adc\_hardware\_average\_mode\_t *mode* )

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

## Parameters

|             |                                                                       |
|-------------|-----------------------------------------------------------------------|
| <i>base</i> | ADC peripheral base address.                                          |
| <i>mode</i> | Setting the hardware average mode. See "adc_hardware_average_mode_t". |

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

## Parameters

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

## Returns

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

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

## Parameters

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

|             |                                                             |
|-------------|-------------------------------------------------------------|
| <i>mask</i> | Mask value for the cleared flags. See "adc_status_flags_t". |
|-------------|-------------------------------------------------------------|

## Chapter 8

# ADC\_ETC: ADC External Trigger Control

### 8.1 Overview

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

### 8.2 Typical use case

#### 8.2.1 Software trigger Configuration

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

#### 8.2.2 Hardware trigger Configuration

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

### Data Structures

- struct `adc_etc_config_t`  
*ADC\_ETC configuration. [More...](#)*
- struct `adc_etc_trigger_chain_config_t`  
*ADC\_ETC trigger chain configuration. [More...](#)*
- struct `adc_etc_trigger_config_t`  
*ADC\_ETC trigger configuration. [More...](#)*

### Macros

- #define `FSL_ADC_ETC_DRIVER_VERSION` (`MAKE_VERSION(2, 2, 1)`)  
*ADC\_ETC driver version.*
- #define `ADC_ETC_DMA_CTRL_TRGn_REQ_MASK` `0xFF0000U`  
*The mask of status flags cleared by writing 1.*

### Enumerations

- enum `_adc_etc_status_flag_mask`  
*ADC\_ETC customized status flags mask.*
- enum `adc_etc_external_trigger_source_t`  
*External triggers sources.*
- enum `adc_etc_interrupt_enable_t`  
*Interrupt enable/disable mask.*

## Initialization

- void `ADC_ETC_Init` (ADC\_ETC\_Type \*base, const `adc_etc_config_t` \*config)  
*Initialize the ADC\_ETC module.*
- void `ADC_ETC_Deinit` (ADC\_ETC\_Type \*base)  
*De-Initialize the ADC\_ETC module.*
- void `ADC_ETC_GetDefaultConfig` (`adc_etc_config_t` \*config)  
*Gets an available pre-defined settings for the ADC\_ETC's configuration.*
- void `ADC_ETC_SetTriggerConfig` (ADC\_ETC\_Type \*base, uint32\_t triggerGroup, const `adc_etc_trigger_config_t` \*config)  
*Set the external XBAR trigger configuration.*
- void `ADC_ETC_SetTriggerChainConfig` (ADC\_ETC\_Type \*base, uint32\_t triggerGroup, uint32\_t chainGroup, const `adc_etc_trigger_chain_config_t` \*config)  
*Set the external XBAR trigger chain configuration.*
- uint32\_t `ADC_ETC_GetInterruptStatusFlags` (ADC\_ETC\_Type \*base, `adc_etc_external_trigger_source_t` sourceIndex)  
*Gets the interrupt status flags of external XBAR and TSC triggers.*
- void `ADC_ETC_ClearInterruptStatusFlags` (ADC\_ETC\_Type \*base, `adc_etc_external_trigger_source_t` sourceIndex, uint32\_t mask)  
*Clears the ADC\_ETC's interrupt status flags.*
- static void `ADC_ETC_EnableDMA` (ADC\_ETC\_Type \*base, uint32\_t triggerGroup)  
*Enable the DMA corresponding to each trigger source.*
- static void `ADC_ETC_DisableDMA` (ADC\_ETC\_Type \*base, uint32\_t triggerGroup)  
*Disable the DMA corresponding to each trigger sources.*
- static uint32\_t `ADC_ETC_GetDMAStatusFlags` (ADC\_ETC\_Type \*base)  
*Get the DMA request status flags.*
- static void `ADC_ETC_ClearDMAStatusFlags` (ADC\_ETC\_Type \*base, uint32\_t mask)  
*Clear the DMA request status flags.*
- static void `ADC_ETC_DoSoftwareReset` (ADC\_ETC\_Type \*base, bool enable)  
*When enable, all logical will be reset.*
- static void `ADC_ETC_DoSoftwareTrigger` (ADC\_ETC\_Type \*base, uint32\_t triggerGroup)  
*Do software trigger corresponding to each XBAR trigger sources.*
- uint32\_t `ADC_ETC_GetADCCConversionValue` (ADC\_ETC\_Type \*base, uint32\_t triggerGroup, uint32\_t chainGroup)  
*Get ADC conversion result from external XBAR sources.*

## 8.3 Data Structure Documentation

### 8.3.1 struct `adc_etc_config_t`

### 8.3.2 struct `adc_etc_trigger_chain_config_t`

### 8.3.3 struct `adc_etc_trigger_config_t`

## 8.4 Macro Definition Documentation

### 8.4.1 #define `FSL_ADC_ETC_DRIVER_VERSION` (MAKE\_VERSION(2, 2, 1))

Version 2.2.1.



#### 8.4.2 #define ADC\_ETC\_DMA\_CTRL\_TRGn\_REQ\_MASK 0xFF0000U

### 8.5 Function Documentation

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

Parameters

|               |                                          |
|---------------|------------------------------------------|
| <i>base</i>   | ADC_ETC peripheral base address.         |
| <i>config</i> | Pointer to "adc_etc_config_t" structure. |

#### 8.5.2 void ADC\_ETC\_Deinit ( ADC\_ETC\_Type \* *base* )

Parameters

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

#### 8.5.3 void ADC\_ETC\_GetDefaultConfig ( adc\_etc\_config\_t \* *config* )

This function initializes the ADC\_ETC's configuration structure with available settings. The default values are:

```
* config->enableTSCBypass = true;
* config->enableTSC0Trigger = false;
* config->enableTSC1Trigger = false;
* config->TSC0triggerPriority = 0U;
* config->TSC1triggerPriority = 0U;
* config->clockPreDivider = 0U;
* config->XBARtriggerMask = 0U;
*
```

Parameters

|               |                                          |
|---------------|------------------------------------------|
| <i>config</i> | Pointer to "adc_etc_config_t" structure. |
|---------------|------------------------------------------|

#### 8.5.4 void ADC\_ETC\_SetTriggerConfig ( ADC\_ETC\_Type \* *base*, uint32\_t *triggerGroup*, const adc\_etc\_trigger\_config\_t \* *config* )

## Parameters

|                     |                                                  |
|---------------------|--------------------------------------------------|
| <i>base</i>         | ADC_ETC peripheral base address.                 |
| <i>triggerGroup</i> | Trigger group index.                             |
| <i>config</i>       | Pointer to "adc_etc_trigger_config_t" structure. |

### 8.5.5 void ADC\_ETC\_SetTriggerChainConfig ( ADC\_ETC\_Type \* *base*, uint32\_t *triggerGroup*, uint32\_t *chainGroup*, const adc\_etc\_trigger\_chain\_config\_t \* *config* )

For example, if *triggerGroup* is set to 0U and *chainGroup* is set to 1U, which means Trigger0 source's chain1 would be configured.

## Parameters

|                     |                                                        |
|---------------------|--------------------------------------------------------|
| <i>base</i>         | ADC_ETC peripheral base address.                       |
| <i>triggerGroup</i> | Trigger group index. Available number is 0~7.          |
| <i>chainGroup</i>   | Trigger chain group index. Available number is 0~7.    |
| <i>config</i>       | Pointer to "adc_etc_trigger_chain_config_t" structure. |

### 8.5.6 uint32\_t ADC\_ETC\_GetInterruptStatusFlags ( ADC\_ETC\_Type \* *base*, adc\_etc\_external\_trigger\_source\_t *sourceIndex* )

## Parameters

|                    |                                  |
|--------------------|----------------------------------|
| <i>base</i>        | ADC_ETC peripheral base address. |
| <i>sourceIndex</i> | trigger source index.            |

## Returns

Status flags mask of trigger. Refer to "\_adc\_etc\_status\_flag\_mask".

### 8.5.7 void ADC\_ETC\_ClearInterruptStatusFlags ( ADC\_ETC\_Type \* *base*, adc\_etc\_external\_trigger\_source\_t *sourceIndex*, uint32\_t *mask* )

## Parameters

|                    |                                                                     |
|--------------------|---------------------------------------------------------------------|
| <i>base</i>        | ADC_ETC peripheral base address.                                    |
| <i>sourceIndex</i> | trigger source index.                                               |
| <i>mask</i>        | Status flags mask of trigger. Refer to "_adc_etc_status_flag_mask". |

### 8.5.8 static void ADC\_ETC\_EnableDMA ( ADC\_ETC\_Type \* *base*, uint32\_t *triggerGroup* ) [inline], [static]

## Parameters

|                     |                                               |
|---------------------|-----------------------------------------------|
| <i>base</i>         | ADC_ETC peripheral base address.              |
| <i>triggerGroup</i> | Trigger group index. Available number is 0~7. |

### 8.5.9 static void ADC\_ETC\_DisableDMA ( ADC\_ETC\_Type \* *base*, uint32\_t *triggerGroup* ) [inline], [static]

## Parameters

|                     |                                               |
|---------------------|-----------------------------------------------|
| <i>base</i>         | ADC_ETC peripheral base address.              |
| <i>triggerGroup</i> | Trigger group index. Available number is 0~7. |

### 8.5.10 static uint32\_t ADC\_ETC\_GetDMAStatusFlags ( ADC\_ETC\_Type \* *base* ) [inline], [static]

Only external XBAR sources support DMA request.

## Parameters

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

## Returns

Mask of external XBAR trigger's DMA request asserted flags. Available range is trigger0:0x01 to trigger7:0x80.

**8.5.11** `static void ADC_ETC_ClearDMAStatusFlags ( ADC_ETC_Type * base,  
uint32_t mask ) [inline], [static]`

Only external XBAR sources support DMA request.

Parameters

|             |                                                                                                                |
|-------------|----------------------------------------------------------------------------------------------------------------|
| <i>base</i> | ADC_ETC peripheral base address.                                                                               |
| <i>mask</i> | Mask of external XBAR trigger's DMA request asserted flags. Available range is trigger0:0x01 to trigger7:0x80. |

### 8.5.12 static void ADC\_ETC\_DoSoftwareReset ( ADC\_ETC\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | ADC_ETC peripheral base address.   |
| <i>enable</i> | Enable/Disable the software reset. |

### 8.5.13 static void ADC\_ETC\_DoSoftwareTrigger ( ADC\_ETC\_Type \* *base*, uint32\_t *triggerGroup* ) [inline], [static]

Each XBAR trigger sources can be configured as HW or SW trigger mode. In hardware trigger mode, trigger source is from XBAR. In software mode, trigger source is from software trigger. TSC trigger sources can only work in hardware trigger mode.

Parameters

|                     |                                               |
|---------------------|-----------------------------------------------|
| <i>base</i>         | ADC_ETC peripheral base address.              |
| <i>triggerGroup</i> | Trigger group index. Available number is 0~7. |

### 8.5.14 uint32\_t ADC\_ETC\_GetADCConversionValue ( ADC\_ETC\_Type \* *base*, uint32\_t *triggerGroup*, uint32\_t *chainGroup* )

For example, if *triggerGroup* is set to 0U and *chainGroup* is set to 1U, which means the API would return Trigger0 source's chain1 conversion result.

Parameters

---

|                     |                                                     |
|---------------------|-----------------------------------------------------|
| <i>base</i>         | ADC_ETC peripheral base address.                    |
| <i>triggerGroup</i> | Trigger group index. Available number is 0~7.       |
| <i>chainGroup</i>   | Trigger chain group index. Available number is 0~7. |

Returns

ADC conversion result value.

# Chapter 9

## AIPSTZ: AHB to IP Bridge

### 9.1 Overview

The MCUXpresso SDK provides a driver for the AHB-to-IP Bridge (AIPSTZ) of MCUXpresso SDK devices.

### Enumerations

- enum `aipstz_master_privilege_level_t` {  
    `kAIPSTZ_MasterBufferedWriteEnable` = (1U << 3),  
    `kAIPSTZ_MasterTrustedForReadEnable` = (1U << 2),  
    `kAIPSTZ_MasterTrustedForWriteEnable` = (1U << 1),  
    `kAIPSTZ_MasterForceUserModeEnable` = 1U }  
    *List of AIPSTZ privilege configuration.*
- enum `aipstz_master_t`  
    *List of AIPSTZ masters.*
- enum `aipstz_peripheral_access_control_t`  
    *List of AIPSTZ peripheral access control configuration.*
- enum `aipstz_peripheral_t`  
    *List of AIPSTZ peripherals.*

### Driver version

- #define `FSL_AIPSTZ_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 1))  
    *Version 2.0.1.*

### Initialization and deinitialization

- void `AIPSTZ_SetMasterPrivilegeLevel` (AIPSTZ\_Type \*base, `aipstz_master_t` master, uint32\_t privilegeConfig)  
    *Configure the privilege level for master.*
- void `AIPSTZ_SetPeripheralAccessControl` (AIPSTZ\_Type \*base, `aipstz_peripheral_t` peripheral, uint32\_t accessControl)  
    *Configure the access for peripheral.*

## 9.2 Enumeration Type Documentation

### 9.2.1 enum `aipstz_master_privilege_level_t`

Enumerator

**`kAIPSTZ_MasterBufferedWriteEnable`** Write accesses from this master are allowed to be buffered.

***kAIPSTZ\_MasterTrustedForReadEnable*** This master is trusted for read accesses.  
***kAIPSTZ\_MasterTrustedForWriteEnable*** This master is trusted for write accesses.  
***kAIPSTZ\_MasterForceUserModeEnable*** Accesses from this master are forced to user-mode.

## 9.2.2 enum aipstz\_master\_t

Organized by width for the 8-15 bits and shift for lower 8 bits.

## 9.2.3 enum aipstz\_peripheral\_access\_control\_t

## 9.2.4 enum aipstz\_peripheral\_t

Organized by register offset for higher 32 bits, width for the 8-15 bits and shift for lower 8 bits.

## 9.3 Function Documentation

### 9.3.1 void AIPSTZ\_SetMasterPriviledgeLevel ( AIPSTZ\_Type \* *base*, aipstz\_master\_t *master*, uint32\_t *privilegeConfig* )

Parameters

|                        |                                                                              |
|------------------------|------------------------------------------------------------------------------|
| <i>base</i>            | AIPSTZ peripheral base pointer                                               |
| <i>master</i>          | Masters for AIPSTZ.                                                          |
| <i>privilegeConfig</i> | Configuration is ORed from <a href="#">aipstz_master_privilege_level_t</a> . |

### 9.3.2 void AIPSTZ\_SetPeripheralAccessControl ( AIPSTZ\_Type \* *base*, aipstz\_peripheral\_t *peripheral*, uint32\_t *accessControl* )

Parameters

|                   |                                |
|-------------------|--------------------------------|
| <i>base</i>       | AIPSTZ peripheral base pointer |
| <i>peripheral</i> | Peripheral for AIPSTZ.         |



|                      |                                                                                 |
|----------------------|---------------------------------------------------------------------------------|
| <i>accessControl</i> | Configuration is ORed from <a href="#">aipstz_peripheral_access_control_t</a> . |
|----------------------|---------------------------------------------------------------------------------|

## Chapter 10

# AOI: Crossbar AND/OR/INVERT Driver

### 10.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Crossbar AND/OR/INVERT (AOI) block of MCUXpresso SDK devices.

The AOI module supports a configurable number of event outputs, where each event output represents a user-programmed combinational boolean function based on four event inputs. The key features of this module include:

- Four dedicated inputs for each event output
- User-programmable combinational boolean function evaluation for each event output
- Memory-mapped device connected to a slave peripheral (IPS) bus
- Configurable number of event outputs

### 10.2 Function groups

#### 10.2.1 AOI Initialization

To initialize the AOI driver, call the [AOI\\_Init\(\)](#) function and pass a baseaddr pointer.

See the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/aoi`.

#### 10.2.2 AOI Get Set Operation

The AOI module provides a universal boolean function generator using a four-term sum of products expression with each product term containing true or complement values of the four selected event inputs (A, B, C, D). The AOI is a highly programmable module for creating combinational boolean outputs for use as hardware triggers. Each selected input term in each product term can be configured to produce a logical 0 or 1 or pass the true or complement of the selected event input. To configure the selected AOI module event, call the API of the [AOI\\_SetEventLogicConfig\(\)](#) function. To get the current event state configure, call the API of [AOI\\_GetEventLogicConfig\(\)](#) function. The AOI module does not support any special modes of operation. See the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/aoi`.

### 10.3 Typical use case

The AOI module is designed to be integrated in conjunction with one or more inter-peripheral crossbar switch (XBAR) modules. A crossbar switch is typically used to select the 4\*n AOI inputs from among available peripheral outputs and GPIO signals. The n EVENTn outputs from the AOI module are typically

used as additional inputs to a second crossbar switch, adding to it the ability to connect to its outputs an arbitrary 4-input boolean function of its other inputs.

This is an example to initialize and configure the AOI driver for a possible use case. Because the AOI module function is directly connected with an XBAR (Inter-peripheral crossbar) module, other peripheral drivers (PIT, CMP, and XBAR) are used to show full functionality of AOI module.

For example: Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver-examples/aoi`

## Data Structures

- struct `aoi_event_config_t`  
*AOI event configuration structure. [More...](#)*

## Macros

- #define `AOI_AOI0`  
*AOI peripheral address.*

## Enumerations

- enum `aoi_input_config_t` {  
    `kAOI_LogicZero` = 0x0U,  
    `kAOI_InputSignal` = 0x1U,  
    `kAOI_InvInputSignal` = 0x2U,  
    `kAOI_LogicOne` = 0x3U }  
*AOI input configurations.*
- enum `aoi_event_t` {  
    `kAOI_Event0` = 0x0U,  
    `kAOI_Event1` = 0x1U,  
    `kAOI_Event2` = 0x2U,  
    `kAOI_Event3` = 0x3U }  
*AOI event indexes, where an event is the collection of the four product terms (0, 1, 2, and 3) and the four signal inputs (A, B, C, and D).*

## Driver version

- #define `FSL_AOI_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 1)`)  
*Version 2.0.1.*

## AOI Initialization

- void `AOI_Init` (`AOI_Type *base`)  
*Initializes an AOI instance for operation.*
- void `AOI_Deinit` (`AOI_Type *base`)  
*Deinitializes an AOI instance for operation.*

## AOI Get Set Operation

- void [AOI\\_GetEventLogicConfig](#) (AOI\_Type \*base, [aoi\\_event\\_t](#) event, [aoi\\_event\\_config\\_t](#) \*config)  
*Gets the Boolean evaluation associated.*
- void [AOI\\_SetEventLogicConfig](#) (AOI\_Type \*base, [aoi\\_event\\_t](#) event, const [aoi\\_event\\_config\\_t](#) \*eventConfig)  
*Configures an AOI event.*

## 10.4 Data Structure Documentation

### 10.4.1 struct [aoi\\_event\\_config\\_t](#)

Defines structure [\\_aoi\\_event\\_config](#) and use the [AOI\\_SetEventLogicConfig\(\)](#) function to make whole event configuration.

### Data Fields

- [aoi\\_input\\_config\\_t](#) PT0AC  
*Product term 0 input A.*
- [aoi\\_input\\_config\\_t](#) PT0BC  
*Product term 0 input B.*
- [aoi\\_input\\_config\\_t](#) PT0CC  
*Product term 0 input C.*
- [aoi\\_input\\_config\\_t](#) PT0DC  
*Product term 0 input D.*
- [aoi\\_input\\_config\\_t](#) PT1AC  
*Product term 1 input A.*
- [aoi\\_input\\_config\\_t](#) PT1BC  
*Product term 1 input B.*
- [aoi\\_input\\_config\\_t](#) PT1CC  
*Product term 1 input C.*
- [aoi\\_input\\_config\\_t](#) PT1DC  
*Product term 1 input D.*
- [aoi\\_input\\_config\\_t](#) PT2AC  
*Product term 2 input A.*
- [aoi\\_input\\_config\\_t](#) PT2BC  
*Product term 2 input B.*
- [aoi\\_input\\_config\\_t](#) PT2CC  
*Product term 2 input C.*
- [aoi\\_input\\_config\\_t](#) PT2DC  
*Product term 2 input D.*
- [aoi\\_input\\_config\\_t](#) PT3AC  
*Product term 3 input A.*
- [aoi\\_input\\_config\\_t](#) PT3BC  
*Product term 3 input B.*
- [aoi\\_input\\_config\\_t](#) PT3CC  
*Product term 3 input C.*
- [aoi\\_input\\_config\\_t](#) PT3DC  
*Product term 3 input D.*

## 10.5 Macro Definition Documentation

### 10.5.1 #define FSL\_AOI\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

## 10.6 Enumeration Type Documentation

### 10.6.1 enum aoi\_input\_config\_t

The selection item represents the Boolean evaluations.

Enumerator

- kAOI\_LogicZero* Forces the input to logical zero.
- kAOI\_InputSignal* Passes the input signal.
- kAOI\_InvInputSignal* Inverts the input signal.
- kAOI\_LogicOne* Forces the input to logical one.

### 10.6.2 enum aoi\_event\_t

Enumerator

- kAOI\_Event0* Event 0 index.
- kAOI\_Event1* Event 1 index.
- kAOI\_Event2* Event 2 index.
- kAOI\_Event3* Event 3 index.

## 10.7 Function Documentation

### 10.7.1 void AOI\_Init ( AOI\_Type \* *base* )

This function un-gates the AOI clock.

Parameters

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

### 10.7.2 void AOI\_Deinit ( AOI\_Type \* *base* )

This function shutdowns AOI module.

## Parameters

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

### 10.7.3 void AOI\_GetEventLogicConfig ( AOI\_Type \* *base*, aoi\_event\_t *event*, aoi\_event\_config\_t \* *config* )

This function returns the Boolean evaluation associated.

Example:

```
aoi_event_config_t demoEventLogicStruct;
AOI_GetEventLogicConfig(AOI, kAOI_Event0, &demoEventLogicStruct);
```

## Parameters

|               |                                                           |
|---------------|-----------------------------------------------------------|
| <i>base</i>   | AOI peripheral address.                                   |
| <i>event</i>  | Index of the event which will be set of type aoi_event_t. |
| <i>config</i> | Selected input configuration .                            |

### 10.7.4 void AOI\_SetEventLogicConfig ( AOI\_Type \* *base*, aoi\_event\_t *event*, const aoi\_event\_config\_t \* *eventConfig* )

This function configures an AOI event according to the aoiEventConfig structure. This function configures all inputs (A, B, C, and D) of all product terms (0, 1, 2, and 3) of a desired event.

Example:

```
aoi_event_config_t demoEventLogicStruct;

demoEventLogicStruct.PT0AC = kAOI_InvInputSignal;
demoEventLogicStruct.PT0BC = kAOI_InputSignal;
demoEventLogicStruct.PT0CC = kAOI_LogicOne;
demoEventLogicStruct.PT0DC = kAOI_LogicOne;

demoEventLogicStruct.PT1AC = kAOI_LogicZero;
demoEventLogicStruct.PT1BC = kAOI_LogicOne;
demoEventLogicStruct.PT1CC = kAOI_LogicOne;
demoEventLogicStruct.PT1DC = kAOI_LogicOne;

demoEventLogicStruct.PT2AC = kAOI_LogicZero;
demoEventLogicStruct.PT2BC = kAOI_LogicOne;
demoEventLogicStruct.PT2CC = kAOI_LogicOne;
demoEventLogicStruct.PT2DC = kAOI_LogicOne;

demoEventLogicStruct.PT3AC = kAOI_LogicZero;
demoEventLogicStruct.PT3BC = kAOI_LogicOne;
demoEventLogicStruct.PT3CC = kAOI_LogicOne;
```

```
demoEventLogicStruct.PT3DC = kAOI_LogicOne;
AOI_SetEventLogicConfig(AOI, kAOI_Event0, demoEventLogicStruct);
```

### Parameters

|                    |                                                                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | AOI peripheral address.                                                                                                                                                    |
| <i>event</i>       | Event which will be configured of type <code>aoi_event_t</code> .                                                                                                          |
| <i>eventConfig</i> | Pointer to type <code>aoi_event_config_t</code> structure. The user is responsible for filling out the members of this structure and passing the pointer to this function. |

# Chapter 11

## BEE: Bus Encryption Engine

### 11.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Bus Encryption Engine (BEE) module.

The BEE module is implemented as an on-the-fly decryption engine. The main features of the BEE module are:

- Standard AXI interconnection
- On-the-fly AES-128 decryption, supporting ECB and CTR mode
- Aliased memory space support. Address remapping for up to two individual regions
- Independent AES Key management for those two individual regions
- Bus access pattern optimization with the aid of local store and forward buffer
- Non-secured access filtering based on security label of the access
- Illegal access check and filtering.

The known hardware limitations of the BEE module are as follows:

- Only supports 128 bits data width AXI interconnection
- Only supports 16-byte burst access size. For a single transaction, the minimum supported access size is limited to 4 bytes.
- Granularity of the address bias is 128 KB per step

### 11.2 BEE Driver Initialization and Configuration

The function [BEE\\_Init\(\)](#) initializes the BEE to default values. The function [BEE\\_GetDefaultConfig\(\)](#) loads default values to the BEE configuration structure. The default values are described below.

See the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/bee. The function [BEE\\_Deinit\(\)](#) performs a hardware reset of BEE module and disables clocks. Configuration and keys from software for both regions are cleared.

### 11.3 Enable & Disable BEE

The function [BEE\\_Enable\(\)](#) enables decryption using BEE. The function [BEE\\_Disable\(\)](#) disables decryption using BEE.

### 11.4 Set BEE region config and key

The function [BEE\\_SetConfig\(\)](#) sets BEE settings according to given configuration structure. The structure is described below.

See the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/bee. The function [BEE\\_SetRegionKey\(\)](#) loads given AES key to BEE register for the given region. The key must be 32-bit aligned and stored in little-endian format. Note that eFuse BEE\_KEYx\_SEL must be set



accordingly to be able to load and use the key loaded in BEE registers. Otherwise, the key cannot be loaded and BEE uses the key from OTPMK or SW\_GP2.

The function [BEE\\_SetRegionNonce\(\)](#) loads given AES nonce (used only for AES CTR mode) to BEE register for the given region. The nonce must be 32-bit aligned and stored in little-endian format.

## 11.5 Status

Provides functions to get and clear the BEE status.

The function [BEE\\_GetStatusFlags\(\)](#) returns status of BEE peripheral. The function [BEE\\_ClearStatusFlags\(\)](#) clears the BEE status flags.

### 11.5.1 BEE example

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

## Data Structures

- struct [bee\\_region\\_config\\_t](#)  
*BEE region configuration structure. [More...](#)*

## Enumerations

- enum [bee\\_aes\\_mode\\_t](#) {  
  [kBEE\\_AesEcbMode](#) = 0U,  
  [kBEE\\_AesCtrMode](#) = 1U }  
*BEE aes mode.*
- enum [bee\\_region\\_t](#) {  
  [kBEE\\_Region0](#) = 0U,  
  [kBEE\\_Region1](#) = 1U }  
*BEE region.*
- enum [bee\\_ac\\_prot\\_enable](#) {  
  [kBEE\\_AccessProtDisabled](#) = 0U,  
  [kBEE\\_AccessProtEnabled](#) = 1U }  
*BEE ac prot enable.*
- enum [bee\\_endian\\_swap\\_enable](#) {  
  [kBEE\\_EndianSwapDisabled](#) = 1U,  
  [kBEE\\_EndianSwapEnabled](#) = 0U }  
*BEE endian swap enable.*
- enum [bee\\_security\\_level](#) {  
  [kBEE\\_SecurityLevel0](#) = 0U,  
  [kBEE\\_SecurityLevel1](#) = 1U,  
  [kBEE\\_SecurityLevel2](#) = 2U,  
  [kBEE\\_SecurityLevel3](#) = 3U }  
*BEE security level.*

- enum `bee_status_flags_t` {  
`kBEE_DisableAbortFlag` = 1U,  
`kBEE_Reg0ReadSecViolation` = 2U,  
`kBEE_ReadIllegalAccess` = 4U,  
`kBEE_Reg1ReadSecViolation` = 8U,  
`kBEE_Reg0AccessViolation` = 16U,  
`kBEE_Reg1AccessViolation` = 32U,  
`kBEE_IdleFlag` = `BEE_STATUS_BEE_IDLE_MASK` }  
*BEE status flags.*

## Functions

- void `BEE_Init` (`BEE_Type *base`)  
*Resets BEE module to factory default values.*
- void `BEE_Deinit` (`BEE_Type *base`)  
*Resets BEE module, clears keys for both regions and disables clock to the BEE.*
- static void `BEE_Enable` (`BEE_Type *base`)  
*Enables BEE decryption.*
- static void `BEE_Disable` (`BEE_Type *base`)  
*Disables BEE decryption.*
- void `BEE_GetDefaultConfig` (`bee_region_config_t *config`)  
*Loads default values to the BEE region configuration structure.*
- void `BEE_SetConfig` (`BEE_Type *base`, const `bee_region_config_t *config`)  
*Sets BEE configuration.*
- `status_t BEE_SetRegionKey` (`BEE_Type *base`, `bee_region_t region`, const `uint8_t *key`, `size_t keySize`)  
*Loads the AES key for selected region into BEE key registers.*
- `status_t BEE_SetRegionNonce` (`BEE_Type *base`, `bee_region_t region`, const `uint8_t *nonce`, `size_t nonceSize`)  
*Loads the nonce for selected region into BEE nonce registers.*
- `uint32_t BEE_GetStatusFlags` (`BEE_Type *base`)  
*Gets the BEE status flags.*
- void `BEE_ClearStatusFlags` (`BEE_Type *base`, `uint32_t mask`)  
*Clears the BEE status flags.*

## Driver version

- `#define FSL_BEE_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`  
*BEE driver version.*

## 11.6 Data Structure Documentation

### 11.6.1 struct `bee_region_config_t`

#### Data Fields

- `bee_aes_mode_t region0Mode`  
*AES mode used for encryption/decryption for region 0.*

- [bee\\_aes\\_mode\\_t region1Mode](#)  
*AES mode used for encryption/decryption for region 1.*
- [uint32\\_t region0AddrOffset](#)  
*Region 0 address offset.*
- [uint32\\_t region1AddrOffset](#)  
*Region 1 address offset.*
- [bee\\_security\\_level region0SecLevel](#)  
*Region 0 security level.*
- [bee\\_security\\_level region1SecLevel](#)  
*Region 1 security level.*
- [uint32\\_t region1Bot](#)  
*Region 1 bottom address.*
- [uint32\\_t region1Top](#)  
*Region 1 top address.*
- [bee\\_ac\\_prot\\_enable accessPermission](#)  
*Access permission control enable/disable.*
- [bee\\_endian\\_swap\\_enable endianSwapEn](#)  
*Endian swap enable/disable.*

## 11.7 Macro Definition Documentation

### 11.7.1 #define FSL\_BEE\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))

Version 2.0.2.

Current version: 2.0.2

Change log:

- 2.0.2
  - Bug Fixes
    - \* Fixed MISRA issue.
- 2.0.1
  - Bug Fixes
    - \* Fixed bug in key user key loading sequence. BEE must be enabled during loading of user key.
    - \* Fixed typos in comments.
  - New Features
    - \* Added configuration setting for endian swap, access permission and region security level.
  - Improvements
    - \* Setting of AES nonce was moved from [BEE\\_SetRegionKey\(\)](#) into separate [BEE\\_SetRegionNonce\(\)](#) function.
      - Changed handling of region settings. Both regions are configured simultaneously by [BEE\\_SetConfig\(\)](#) function. Configuration of FAC start and end address using IOM-UXC\_GPRs was moved to application.
    - \* Default value for region address offset was changed to 0.
- 2.0.0
  - Initial version

## 11.8 Enumeration Type Documentation

### 11.8.1 enum bee\_aes\_mode\_t

Enumerator

*kBEE\_AesEcbMode* AES ECB Mode.

*kBEE\_AesCtrMode* AES CTR Mode.

### 11.8.2 enum bee\_region\_t

Enumerator

*kBEE\_Region0* BEE region 0.

*kBEE\_Region1* BEE region 1.

### 11.8.3 enum bee\_ac\_prot\_enable

Enumerator

*kBEE\_AccessProtDisabled* BEE access permission control disabled.

*kBEE\_AccessProtEnabled* BEE access permission control enabled.

### 11.8.4 enum bee\_endian\_swap\_enable

Enumerator

*kBEE\_EndianSwapDisabled* BEE endian swap disabled.

*kBEE\_EndianSwapEnabled* BEE endian swap enabled.

### 11.8.5 enum bee\_security\_level

Enumerator

*kBEE\_SecurityLevel0* BEE security level 0.

*kBEE\_SecurityLevel1* BEE security level 1.

*kBEE\_SecurityLevel2* BEE security level 2.

*kBEE\_SecurityLevel3* BEE security level 3.

## 11.8.6 enum bee\_status\_flags\_t

Enumerator

*kBEE\_DisableAbortFlag* Disable abort flag.  
*kBEE\_Reg0ReadSecViolation* Region-0 read channel security violation.  
*kBEE\_ReadIllegalAccess* Read channel illegal access detected.  
*kBEE\_Reg1ReadSecViolation* Region-1 read channel security violation.  
*kBEE\_Reg0AccessViolation* Protected region-0 access violation.  
*kBEE\_Reg1AccessViolation* Protected region-1 access violation.  
*kBEE\_IdleFlag* Idle flag.

## 11.9 Function Documentation

### 11.9.1 void BEE\_Init ( BEE\_Type \* *base* )

This function performs hardware reset of BEE module. Attributes and keys from software for both regions are cleared.

Parameters

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

### 11.9.2 void BEE\_Deinit ( BEE\_Type \* *base* )

This function performs hardware reset of BEE module and disables clocks. Attributes and keys from software for both regions are cleared.

Parameters

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

### 11.9.3 static void BEE\_Enable ( BEE\_Type \* *base* ) [inline], [static]

This function enables decryption using BEE.

Parameters

---

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

#### 11.9.4 static void BEE\_Disable ( BEE\_Type \* *base* ) [inline], [static]

This function disables decryption using BEE.

Parameters

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

#### 11.9.5 void BEE\_GetDefaultConfig ( bee\_region\_config\_t \* *config* )

Loads default values to the BEE region configuration structure. The default values are as follows:

```
* config->region0Mode = kBEE_AesCtrMode;
* config->region1Mode = kBEE_AesCtrMode;
* config->region0AddrOffset = 0U;
* config->region1AddrOffset = 0U;
* config->region0SecLevel = kBEE_SecurityLevel3;
* config->region1SecLevel = kBEE_SecurityLevel3;
* config->region1Bot = 0U;
* config->region1Top = 0U;
* config->accessPermission = kBEE_AccessProtDisabled;
* config->endianSwapEn = kBEE_EndianSwapEnabled;
*
```

Parameters

|               |                                             |
|---------------|---------------------------------------------|
| <i>config</i> | Configuration structure for BEE peripheral. |
|---------------|---------------------------------------------|

#### 11.9.6 void BEE\_SetConfig ( BEE\_Type \* *base*, const bee\_region\_config\_t \* *config* )

This function sets BEE peripheral and BEE region settings according to given configuration structure.

Parameters

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

|               |                                  |
|---------------|----------------------------------|
| <i>config</i> | Configuration structure for BEE. |
|---------------|----------------------------------|

### 11.9.7 **status\_t BEE\_SetRegionKey ( BEE\_Type \* *base*, bee\_region\_t *region*, const uint8\_t \* *key*, size\_t *keySize* )**

This function loads given AES key to BEE register for the given region. The key must be 32-bit aligned and stored in little-endian format.

Please note, that eFuse BEE\_KEYx\_SEL must be set accordingly to be able to load and use key loaded in BEE registers. Otherwise, key cannot be loaded and BEE will use key from OTPMK or SW\_GP2.

Parameters

|                |                                               |
|----------------|-----------------------------------------------|
| <i>base</i>    | BEE peripheral address.                       |
| <i>region</i>  | Selection of the BEE region to be configured. |
| <i>key</i>     | AES key (in little-endian format).            |
| <i>keySize</i> | Size of AES key.                              |

### 11.9.8 **status\_t BEE\_SetRegionNonce ( BEE\_Type \* *base*, bee\_region\_t *region*, const uint8\_t \* *nonce*, size\_t *nonceSize* )**

This function loads given nonce (only AES CTR mode) to BEE register for the given region. The nonce must be 32-bit aligned and stored in little-endian format.

Parameters

|                  |                                               |
|------------------|-----------------------------------------------|
| <i>base</i>      | BEE peripheral address.                       |
| <i>region</i>    | Selection of the BEE region to be configured. |
| <i>nonce</i>     | AES nonce (in little-endian format).          |
| <i>nonceSize</i> | Size of AES nonce.                            |

### 11.9.9 **uint32\_t BEE\_GetStatusFlags ( BEE\_Type \* *base* )**

This function returns status of BEE peripheral.

Parameters

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

Returns

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

**11.9.10 void BEE\_ClearStatusFlags ( BEE\_Type \* *base*, uint32\_t *mask* )**

Parameters

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



## Chapter 12

# CACHE: ARMV7-M7 CACHE Memory Controller

### 12.1 Overview

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

The CACHE driver is created to help the user more easily operate the cache memory. The APIs for basic operations are including the following three levels:

1L. The L1 cache driver API. This level provides the level 1 caches controller drivers. The L1 caches are mainly integrated in the Core memory system, Cortex-M7 L1 caches, etc. For our Cortex-M4 series platforms, the L1 cache is the local memory controller (LMEM) which is not integrated in the Cortex-M4 processor memory system.

2L. The L2 cache driver API. This level provides the level 2 cache controller drivers. The L2 cache could be integrated in the CORE memory system or an external L2 cache memory, PL310, etc.

3L. The combined cache driver API. This level provides many APIs for combined L1 and L2 cache maintain operations. This is provided for MCUXpresso SDK drivers (DMA, ENET, USDHC, etc) which should do the cache maintenance in their transactional APIs.

### 12.2 Function groups

#### 12.2.1 L1 CACHE Operation

The L1 CACHE has both code cache and data cache. This function group provides independent two groups API for both code cache and data cache. There are Enable/Disable APIs for code cache and data cache control and cache maintenance operations as Invalidate/Clean/CleanInvalidate by all and by address range.

#### 12.2.2 L2 CACHE Operation

The L2 CACHE does not divide the cache to data and code. Instead, this function group provides one group cache maintenance operations as Enable/Disable/Invalidate/Clean/CleanInvalidate by all and by address range. Except the maintenance operation APIs, the L2 CACHE has it's initialization/configure API. The user can use the default configure parameter by calling `L2CACHE_GetDefaultConfig()` or changing the parameters as they wish. Then, call `L2CACHE_Init` to do the L2 CACHE initialization. After initialization, the L2 cache can then be enabled.

Note: For the core external l2 Cache, the SoC usually has the control bit to select the SRAM to use as L2 Cache or normal SRAM. Make sure this selection is right when you use the L2 CACHE feature.

## Driver version

- #define **FSL\_CACHE\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 4))  
*cache driver version 2.0.4.*

## Control for cortex-m7 L1 cache

- static void **L1CACHE\_EnableICache** (void)  
*Enables cortex-m7 L1 instruction cache.*
- static void **L1CACHE\_DisableICache** (void)  
*Disables cortex-m7 L1 instruction cache.*
- static void **L1CACHE\_InvalidateICache** (void)  
*Invalidate cortex-m7 L1 instruction cache.*
- void **L1CACHE\_InvalidateICacheByRange** (uint32\_t address, uint32\_t size\_byte)  
*Invalidate cortex-m7 L1 instruction cache by range.*
- static void **L1CACHE\_EnableDCache** (void)  
*Enables cortex-m7 L1 data cache.*
- static void **L1CACHE\_DisableDCache** (void)  
*Disables cortex-m7 L1 data cache.*
- static void **L1CACHE\_InvalidateDCache** (void)  
*Invalidates cortex-m7 L1 data cache.*
- static void **L1CACHE\_CleanDCache** (void)  
*Cleans cortex-m7 L1 data cache.*
- static void **L1CACHE\_CleanInvalidateDCache** (void)  
*Cleans and Invalidates cortex-m7 L1 data cache.*
- static void **L1CACHE\_InvalidateDCacheByRange** (uint32\_t address, uint32\_t size\_byte)  
*Invalidates cortex-m7 L1 data cache by range.*
- static void **L1CACHE\_CleanDCacheByRange** (uint32\_t address, uint32\_t size\_byte)  
*Cleans cortex-m7 L1 data cache by range.*
- static void **L1CACHE\_CleanInvalidateDCacheByRange** (uint32\_t address, uint32\_t size\_byte)  
*Cleans and Invalidates cortex-m7 L1 data cache by range.*

## Unified Cache Control for all caches (cortex-m7 L1 cache + I2 pl310)

Mainly used for many drivers for easy cache operation.

- void **ICACHE\_InvalidateByRange** (uint32\_t address, uint32\_t size\_byte)  
*Invalidates all instruction caches by range.*
- void **DCACHE\_InvalidateByRange** (uint32\_t address, uint32\_t size\_byte)  
*Invalidates all data caches by range.*
- void **DCACHE\_CleanByRange** (uint32\_t address, uint32\_t size\_byte)  
*Cleans all data caches by range.*
- void **DCACHE\_CleanInvalidateByRange** (uint32\_t address, uint32\_t size\_byte)  
*Cleans and Invalidates all data caches by range.*

## 12.3 Macro Definition Documentation

### 12.3.1 #define FSL\_CACHE\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 4))

## 12.4 Function Documentation

12.4.1 void L1CACHE\_InvalidateICacheByRange ( uint32\_t *address*, uint32\_t *size\_byte* )

## Parameters

|                  |                                                    |
|------------------|----------------------------------------------------|
| <i>address</i>   | The start address of the memory to be invalidated. |
| <i>size_byte</i> | The memory size.                                   |

## Note

The start address and *size\_byte* should be 32-byte(FSL\_FEATURE\_L1ICACHE\_LINESIZE\_BYTE) aligned. The startAddr here will be forced to align to L1 I-cache line size if startAddr is not aligned. For the *size\_byte*, application should make sure the alignment or make sure the right operation order if the *size\_byte* is not aligned.

#### 12.4.2 static void L1CACHE\_InvalidateDCacheByRange ( uint32\_t *address*, uint32\_t *size\_byte* ) [inline], [static]

## Parameters

|                  |                                                    |
|------------------|----------------------------------------------------|
| <i>address</i>   | The start address of the memory to be invalidated. |
| <i>size_byte</i> | The memory size.                                   |

## Note

The start address and *size\_byte* should be 32-byte(FSL\_FEATURE\_L1DCACHE\_LINESIZE\_BYTE) aligned. The startAddr here will be forced to align to L1 D-cache line size if startAddr is not aligned. For the *size\_byte*, application should make sure the alignment or make sure the right operation order if the *size\_byte* is not aligned.

#### 12.4.3 static void L1CACHE\_CleanDCacheByRange ( uint32\_t *address*, uint32\_t *size\_byte* ) [inline], [static]

## Parameters

|                  |                                                |
|------------------|------------------------------------------------|
| <i>address</i>   | The start address of the memory to be cleaned. |
| <i>size_byte</i> | The memory size.                               |

## Note

The start address and *size\_byte* should be 32-byte(FSL\_FEATURE\_L1DCACHE\_LINESIZE\_BYTE) aligned. The startAddr here will be forced to align to L1 D-cache line size if startAddr is not aligned. For the *size\_byte*, application should make sure the alignment or make sure the right operation order if the *size\_byte* is not aligned.

12.4.4 `static void L1CACHE_CleanInvalidateDCacheByRange ( uint32_t address,  
uint32_t size_byte ) [inline], [static]`

## Parameters

|                  |                                                              |
|------------------|--------------------------------------------------------------|
| <i>address</i>   | The start address of the memory to be clean and invalidated. |
| <i>size_byte</i> | The memory size.                                             |

## Note

The start address and *size\_byte* should be 32-byte(FSL\_FEATURE\_L1DCACHE\_LINESIZE\_BYTE) aligned. The startAddr here will be forced to align to L1 D-cache line size if startAddr is not aligned. For the *size\_byte*, application should make sure the alignment or make sure the right operation order if the *size\_byte* is not aligned.

#### 12.4.5 void ICACHE\_InvalidateByRange ( uint32\_t *address*, uint32\_t *size\_byte* )

Both cortex-m7 L1 cache line and L2 PL310 cache line length is 32-byte.

## Parameters

|                  |                                       |
|------------------|---------------------------------------|
| <i>address</i>   | The physical address.                 |
| <i>size_byte</i> | size of the memory to be invalidated. |

## Note

*address* and *size* should be aligned to cache line size 32-Byte due to the cache operation unit is one cache line. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the *size\_byte*, application should make sure the alignment or make sure the right operation order if the *size\_byte* is not aligned.

#### 12.4.6 void DCACHE\_InvalidateByRange ( uint32\_t *address*, uint32\_t *size\_byte* )

Both cortex-m7 L1 cache line and L2 PL310 cache line length is 32-byte.

## Parameters

|                |                       |
|----------------|-----------------------|
| <i>address</i> | The physical address. |
|----------------|-----------------------|

|                  |                                       |
|------------------|---------------------------------------|
| <i>size_byte</i> | size of the memory to be invalidated. |
|------------------|---------------------------------------|

## Note

address and size should be aligned to cache line size 32-Byte due to the cache operation unit is one cache line. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the *size\_byte*, application should make sure the alignment or make sure the right operation order if the *size\_byte* is not aligned.

#### 12.4.7 void DCACHE\_CleanByRange ( uint32\_t address, uint32\_t size\_byte )

Both cortex-m7 L1 cache line and L2 PL310 cache line length is 32-byte.

## Parameters

|                  |                                   |
|------------------|-----------------------------------|
| <i>address</i>   | The physical address.             |
| <i>size_byte</i> | size of the memory to be cleaned. |

## Note

address and size should be aligned to cache line size 32-Byte due to the cache operation unit is one cache line. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the *size\_byte*, application should make sure the alignment or make sure the right operation order if the *size\_byte* is not aligned.

#### 12.4.8 void DCACHE\_CleanInvalidateByRange ( uint32\_t address, uint32\_t size\_byte )

Both cortex-m7 L1 cache line and L2 PL310 cache line length is 32-byte.

## Parameters

|                  |                                                   |
|------------------|---------------------------------------------------|
| <i>address</i>   | The physical address.                             |
| <i>size_byte</i> | size of the memory to be cleaned and invalidated. |

## Note

address and size should be aligned to cache line size 32-Byte due to the cache operation unit is one cache line. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the *size\_byte*, application should make sure the alignment or make sure the right operation order if the *size\_byte* is not aligned.

# Chapter 13

## CMP: Analog Comparator Driver

### 13.1 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.

### 13.2 Typical use case

#### 13.2.1 Polling Configuration

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

#### 13.2.2 Interrupt Configuration

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

### Data Structures

- struct `cmp_config_t`  
*Configures the comparator. [More...](#)*
- struct `cmp_filter_config_t`  
*Configures the filter. [More...](#)*
- struct `cmp_dac_config_t`  
*Configures the internal DAC. [More...](#)*

### Enumerations

- enum `_cmp_interrupt_enable` {  
    `kCMP_OutputRisingInterruptEnable` = `CMP_SCR_IER_MASK`,  
    `kCMP_OutputFallingInterruptEnable` = `CMP_SCR_IEF_MASK` }  
*Interrupt enable/disable mask.*
- enum `_cmp_status_flags` {  
    `kCMP_OutputRisingEventFlag` = `CMP_SCR_CFR_MASK`,  
    `kCMP_OutputFallingEventFlag` = `CMP_SCR_CFF_MASK`,  
    `kCMP_OutputAssertEventFlag` = `CMP_SCR_COUT_MASK` }  
*Status flags' mask.*



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

## Driver version

- `#define FSL_CMP_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))`  
*CMP driver version 2.0.3.*

## 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_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.*

## 13.3 Data Structure Documentation

### 13.3.1 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.*

#### Field Documentation

- (1) **bool cmp\_config\_t::enableCmp**
- (2) **cmp\_hysteresis\_mode\_t cmp\_config\_t::hysteresisMode**
- (3) **bool cmp\_config\_t::enableHighSpeed**
- (4) **bool cmp\_config\_t::enableInvertOutput**
- (5) **bool cmp\_config\_t::useUnfilteredOutput**
- (6) **bool cmp\_config\_t::enablePinOut**

### 13.3.2 struct cmp\_filter\_config\_t

#### Data Fields

- uint8\_t [filterCount](#)  
*Filter Sample Count.*
- uint8\_t [filterPeriod](#)  
*Filter Sample Period.*

#### Field Documentation

- (1) **uint8\_t cmp\_filter\_config\_t::filterCount**

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

(2) `uint8_t cmp_filter_config_t::filterPeriod`

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

### 13.3.3 struct `cmp_dac_config_t`

#### Data Fields

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

#### Field Documentation

(1) `cmp_reference_voltage_source_t cmp_dac_config_t::referenceVoltageSource`

(2) `uint8_t cmp_dac_config_t::DACValue`

Available range is 0-63.

## 13.4 Macro Definition Documentation

### 13.4.1 #define `FSL_CMP_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))`

## 13.5 Enumeration Type Documentation

### 13.5.1 enum `_cmp_interrupt_enable`

Enumerator

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

### 13.5.2 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.

### 13.5.3 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.

### 13.5.4 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.

## 13.6 Function Documentation

### 13.6.1 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. |

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

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

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

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

## Parameters

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

### 13.6.3 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. |

### 13.6.4 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. |
|---------------|-----------------------------------------|

### 13.6.5 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. |

### 13.6.6 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. |

### 13.6.7 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. |

### 13.6.8 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". |

### 13.6.9 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". |

**13.6.10 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".

**13.6.11 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 14

## Common Driver

### 14.1 Overview

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

#### Macros

- #define `FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ` 1  
*Macro to use the default weak IRQ handler in drivers.*
- #define `MAKE_STATUS`(group, code) (((group)\*100L) + (code))  
*Construct a status code value from a group and code number.*
- #define `MAKE_VERSION`(major, minor, bugfix) (((major)\*65536L) + ((minor)\*256L) + (bugfix))  
*Construct the version number for drivers.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_NONE` 0U  
*No debug console.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_UART` 1U  
*Debug console based on UART.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_LPUART` 2U  
*Debug console based on LPUART.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_LPSCI` 3U  
*Debug console based on LPSCI.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_USBCDC` 4U  
*Debug console based on USBCDC.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM` 5U  
*Debug console based on FLEXCOMM.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_IUART` 6U  
*Debug console based on i.MX UART.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_VUSART` 7U  
*Debug console based on LPC\_VUSART.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART` 8U  
*Debug console based on LPC\_USART.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_SWO` 9U  
*Debug console based on SWO.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_QSCI` 10U  
*Debug console based on QSCI.*
- #define `ARRAY_SIZE`(x) (sizeof(x) / sizeof((x)[0]))  
*Computes the number of elements in an array.*

#### Typedefs

- typedef int32\_t `status_t`  
*Type used for all status and error return values.*



## Enumerations

- enum `_status_groups` {
  - `kStatusGroup_Generic` = 0,
  - `kStatusGroup_FLASH` = 1,
  - `kStatusGroup_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_USMCI` = 57,
  - `kStatusGroup_DMIC` = 58,
  - `kStatusGroup_SDIF` = 59,

```
kStatusGroup_NETC = 166 }
```

*Status group numbers.*

- enum {
  - kStatus\_Success = MAKE\_STATUS(kStatusGroup\_Generic, 0),
  - kStatus\_Fail = MAKE\_STATUS(kStatusGroup\_Generic, 1),
  - kStatus\_ReadOnly = MAKE\_STATUS(kStatusGroup\_Generic, 2),
  - kStatus\_OutOfRange = MAKE\_STATUS(kStatusGroup\_Generic, 3),
  - kStatus\_InvalidArgument = MAKE\_STATUS(kStatusGroup\_Generic, 4),
  - kStatus\_Timeout = MAKE\_STATUS(kStatusGroup\_Generic, 5),
  - kStatus\_NoTransferInProgress,
  - kStatus\_Busy = MAKE\_STATUS(kStatusGroup\_Generic, 7),
  - kStatus\_NoData }

*Generic status return codes.*

## Functions

- void \* **SDK\_Malloc** (size\_t size, size\_t alignbytes)
  - Allocate memory with given alignment and aligned size.*
- void **SDK\_Free** (void \*ptr)
  - Free memory.*
- void **SDK\_DelayAtLeastUs** (uint32\_t delayTime\_us, uint32\_t coreClock\_Hz)
  - Delay at least for some time.*

## Driver version

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

## Min/max macros

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

## UINT16\_MAX/UINT32\_MAX value

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

## Suppress fallthrough warning macro

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

## 14.2 Macro Definition Documentation

14.2.1 **#define FSL\_DRIVER\_TRANSFER\_DOUBLE\_WEAK\_IRQ 1**

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

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

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

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

14.2.4 **#define FSL\_COMMON\_DRIVER\_VERSION (MAKE\_VERSION(2, 4, 0))**

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

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

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

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

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

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

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

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

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

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

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

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

### 14.3 Typedef Documentation

14.3.1 **typedef int32\_t status\_t**

### 14.4 Enumeration Type Documentation

14.4.1 **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\_TOUCH\_PANEL* Group number for touch panel status codes.

*kStatusGroup\_VBAT* Group number for VBAT status codes.

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

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

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

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

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

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

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

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

*kStatusGroup\_HAL\_I2S* Group number for HAL I2S status codes.

*kStatusGroup\_HAL\_ADC\_SENSOR* Group number for HAL ADC SENSOR status codes.

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

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

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

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

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

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

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

*kStatusGroup\_LIST* Group number for List status codes.  
*kStatusGroup\_OSA* Group number for OSA status codes.  
*kStatusGroup\_COMMON\_TASK* Group number for Common task status codes.  
*kStatusGroup\_MSG* Group number for messaging status codes.  
*kStatusGroup\_SDK\_OCOTP* Group number for OCOTP status codes.  
*kStatusGroup\_SDK\_FLEXSPINOR* Group number for FLEXSPINOR status codes.  
*kStatusGroup\_CODEC* Group number for codec status codes.  
*kStatusGroup\_ASRC* Group number for codec status ASRC.  
*kStatusGroup\_OTFAD* Group number for codec status codes.  
*kStatusGroup\_SDIO SLV* Group number for SDIO SLV status codes.  
*kStatusGroup\_MECC* Group number for MECC status codes.  
*kStatusGroup\_ENET\_QOS* Group number for ENET\_QOS status codes.  
*kStatusGroup\_LOG* Group number for LOG status codes.  
*kStatusGroup\_I3CBUS* Group number for I3CBUS status codes.  
*kStatusGroup\_QSCI* Group number for QSCI status codes.  
*kStatusGroup\_SNT* Group number for SNT status codes.  
*kStatusGroup\_QUEUEDSPI* Group number for QSPI status codes.  
*kStatusGroup\_POWER\_MANAGER* Group number for POWER\_MANAGER status codes.  
*kStatusGroup\_IPED* Group number for IPED status codes.  
*kStatusGroup\_ELS\_PKC* Group number for ELS PKC status codes.  
*kStatusGroup\_CSS\_PKC* Group number for CSS PKC status codes.  
*kStatusGroup\_HOSTIF* Group number for HOSTIF status codes.  
*kStatusGroup\_CLIF* Group number for CLIF status codes.  
*kStatusGroup\_BMA* Group number for BMA status codes.  
*kStatusGroup\_NETC* Group number for NETC status codes.

## 14.4.2 anonymous enum

Enumerator

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

## 14.5 Function Documentation

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

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

## Parameters

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

## Return values

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

**14.5.2 void SDK\_Free ( void \* *ptr* )**

## Parameters

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

**14.5.3 void SDK\_DelayAtLeastUs ( uint32\_t *delayTime\_us*, uint32\_t *coreClock\_Hz* )**

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

## Parameters

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



# Chapter 15

## DCDC: DCDC Converter

### 15.1 Overview

The MCUXpresso SDK provides a peripheral driver for the DCDC Converter (DCDC) module of MCU-Xpresso SDK devices.

The DCDC converter module is a synchronous buck mode DCDC converter. It can produce single outputs for SoC peripherals and external devices with high conversion efficiency. The converter can be operated in continuous or pulsed mode.

As a module to provide the power for hardware system, the DCDC starts working when the system is powered up before the software takes over the SoC. Some important configuration is done by the board settings. Before the software can access the DCDC's registers, the DCDC is already working normally with the default settings.

However, if the application needs to improve the DCDC's performance or change the default settings, the DCDC driver would help. The DCDC's register cannot be accessed by software before its initialization (open the clock gate). The user can configure the hardware according to the application guide from reference manual.

### 15.2 Function groups

#### 15.2.1 Initialization and deinitialization

This function group is to enable/disable the operations to DCDC module through the driver.

#### 15.2.2 Status

Provides functions to get the DCDC status.

#### 15.2.3 Misc control

Provides functions to set the DCDC's miscellaneous control.

#### Set point mode control

Provides functions to initialize/de-initialize DCDC module in set point mode.

## 15.3 Application guideline

### 15.3.1 Continuous conduction mode

### 15.3.2 Discontinuous conduction mode

/\*!

## Data Structures

- struct `dcdc_detection_config_t`  
*Configuration for DCDC detection. [More...](#)*
- struct `dcdc_loop_control_config_t`  
*Configuration for the loop control. [More...](#)*
- struct `dcdc_low_power_config_t`  
*Configuration for DCDC low power. [More...](#)*
- struct `dcdc_internal_regulator_config_t`  
*Configuration for DCDC internal regulator. [More...](#)*
- struct `dcdc_min_power_config_t`  
*Configuration for min power setting. [More...](#)*

## Macros

- `#define FSL_DCDC_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))`  
*DCDC driver version.*

## Enumerations

- enum `_dcdc_status_flags_t` { `kDCDC_LockedOKStatus = (1U << 0U)` }  
*DCDC status flags.*
- enum `dcdc_comparator_current_bias_t` {  
`kDCDC_ComparatorCurrentBias50nA = 0U,`  
`kDCDC_ComparatorCurrentBias100nA = 1U,`  
`kDCDC_ComparatorCurrentBias200nA = 2U,`  
`kDCDC_ComparatorCurrentBias400nA = 3U` }  
*The current bias of low power comparator.*
- enum `dcdc_over_current_threshold_t` {  
`kDCDC_OverCurrentThresholdAlt0 = 0U,`  
`kDCDC_OverCurrentThresholdAlt1 = 1U,`  
`kDCDC_OverCurrentThresholdAlt2 = 2U,`  
`kDCDC_OverCurrentThresholdAlt3 = 3U` }  
*The threshold of over current detection.*
- enum `dcdc_peak_current_threshold_t` {  
`kDCDC_PeakCurrentThresholdAlt0 = 0U,`  
`kDCDC_PeakCurrentThresholdAlt1 = 1U,`  
`kDCDC_PeakCurrentThresholdAlt2 = 2U,`  
`kDCDC_PeakCurrentThresholdAlt3 = 3U,`  
`kDCDC_PeakCurrentThresholdAlt4 = 4U,`

`kDCDC_PeakCurrentThresholdAlt5 = 5U }`

*The threshold if peak current detection.*

- enum `dcdc_count_charging_time_period_t` {  
`kDCDC_CountChargingTimePeriod8Cycle = 0U,`  
`kDCDC_CountChargingTimePeriod16Cycle = 1U }`

*The period of counting the charging times in power save mode.*

- enum `dcdc_count_charging_time_threshold_t` {  
`kDCDC_CountChargingTimeThreshold32 = 0U,`  
`kDCDC_CountChargingTimeThreshold64 = 1U,`  
`kDCDC_CountChargingTimeThreshold16 = 2U,`  
`kDCDC_CountChargingTimeThreshold8 = 3U }`

*The threshold of the counting number of charging times.*

- enum `dcdc_clock_source_t` {  
`kDCDC_ClockAutoSwitch = 0U,`  
`kDCDC_ClockInternalOsc = 1U,`  
`kDCDC_ClockExternalOsc = 2U }`

*Oscillator clock option.*

## Initialization and deinitialization

- void `DCDC_Init` (`DCDC_Type *base`)  
*Enable the access to DCDC registers.*
- void `DCDC_Deinit` (`DCDC_Type *base`)  
*Disable the access to DCDC registers.*

## Status

- uint32\_t `DCDC_GetstatusFlags` (`DCDC_Type *base`)  
*Get DCDC status flags.*

## Misc control

- static void `DCDC_EnableOutputRangeComparator` (`DCDC_Type *base`, bool enable)  
*Enable the output range comparator.*
- void `DCDC_SetClockSource` (`DCDC_Type *base`, `dcdc_clock_source_t` clockSource)  
*Configure the DCDC clock source.*
- void `DCDC_GetDefaultDetectionConfig` (`dcdc_detection_config_t *config`)  
*Get the default setting for detection configuration.*
- void `DCDC_SetDetectionConfig` (`DCDC_Type *base`, const `dcdc_detection_config_t *config`)  
*Configure the DCDC detection.*
- void `DCDC_GetDefaultLowPowerConfig` (`dcdc_low_power_config_t *config`)  
*Get the default setting for low power configuration.*
- void `DCDC_SetLowPowerConfig` (`DCDC_Type *base`, const `dcdc_low_power_config_t *config`)  
*Configure the DCDC low power.*
- void `DCDC_ResetCurrentAlertSignal` (`DCDC_Type *base`, bool enable)  
*Reset current alert signal.*
- static void `DCDC_SetBandgapVoltageTrimValue` (`DCDC_Type *base`, uint32\_t trimValue)  
*Set the bangap trim value to trim bandgap voltage.*
- void `DCDC_GetDefaultLoopControlConfig` (`dcdc_loop_control_config_t *config`)

- *Get the default setting for loop control configuration.*
- void [DCDC\\_SetLoopControlConfig](#) (DCDC\_Type \*base, const [dcdc\\_loop\\_control\\_config\\_t](#) \*config)
- *Configure the DCDC loop control.*
- void [DCDC\\_SetMinPowerConfig](#) (DCDC\_Type \*base, const [dcdc\\_min\\_power\\_config\\_t](#) \*config)
- *Configure for the min power.*
- static void [DCDC\\_SetLPComparatorBiasValue](#) (DCDC\_Type \*base, [dcdc\\_comparator\\_current\\_bias\\_t](#) biasVaule)
- *Set the current bias of low power comparator.*
- static void [DCDC\\_LockTargetVoltage](#) (DCDC\_Type \*base)
- *Lock target voltage.*
- void [DCDC\\_AdjustTargetVoltage](#) (DCDC\_Type \*base, uint32\_t VDDRun, uint32\_t VDDStandby)
- *Adjust the target voltage of VDD\_SOC in run mode and low power mode.*
- void [DCDC\\_AdjustRunTargetVoltage](#) (DCDC\_Type \*base, uint32\_t VDDRun)
- *Adjust the target voltage of VDD\_SOC in run mode.*
- void [DCDC\\_AdjustLowPowerTargetVoltage](#) (DCDC\_Type \*base, uint32\_t VDDStandby)
- *Adjust the target voltage of VDD\_SOC in low power mode.*
- void [DCDC\\_SetInternalRegulatorConfig](#) (DCDC\_Type \*base, const [dcdc\\_internal\\_regulator\\_config\\_t](#) \*config)
- *Configure the DCDC internal regulator.*
- static void [DCDC\\_EnableImproveTransition](#) (DCDC\_Type \*base, bool enable)
- *Enable/Disable to improve the transition from heavy load to light load.*

## Application guideline

- void [DCDC\\_BootIntoDCM](#) (DCDC\_Type \*base)
- *Boot DCDC into DCM(discontinuous conduction mode).*
- void [DCDC\\_BootIntoCCM](#) (DCDC\_Type \*base)
- *Boot DCDC into CCM(continous conduction mode).*

## 15.4 Data Structure Documentation

### 15.4.1 struct [dcdc\\_detection\\_config\\_t](#)

#### Data Fields

- bool [enableXtalokDetection](#)
- *Enable xtalok detection circuit.*
- bool [powerDownOverVoltageDetection](#)
- *Power down over-voltage detection comparator.*
- bool [powerDownLowVlotageDetection](#)
- *Power down low-voltage detection comparator.*
- bool [powerDownOverCurrentDetection](#)
- *Power down over-current detection.*
- bool [powerDownPeakCurrentDetection](#)
- *Power down peak-current detection.*
- bool [powerDownZeroCrossDetection](#)
- *Power down the zero cross detection function for discontinuous conductor mode.*
- [dcdc\\_over\\_current\\_threshold\\_t](#) [OverCurrentThreshold](#)
- *The threshold of over current detection.*

- [dcdc\\_peak\\_current\\_threshold\\_t PeakCurrentThreshold](#)  
*The threshold of peak current detection.*

#### Field Documentation

- (1) `bool dcdc_detection_config_t::enableXtalokDetection`
- (2) `bool dcdc_detection_config_t::powerDownOverVoltageDetection`
- (3) `bool dcdc_detection_config_t::powerDownLowVoltageDetection`
- (4) `bool dcdc_detection_config_t::powerDownOverCurrentDetection`
- (5) `bool dcdc_detection_config_t::powerDownPeakCurrentDetection`
- (6) `bool dcdc_detection_config_t::powerDownZeroCrossDetection`
- (7) `dc_dc_over_current_threshold_t dcdc_detection_config_t::OverCurrentThreshold`
- (8) `dc_dc_peak_current_threshold_t dcdc_detection_config_t::PeakCurrentThreshold`

### 15.4.2 struct dcdc\_loop\_control\_config\_t

#### Data Fields

- `bool enableCommonHysteresis`  
*Enable hysteresis in switching converter common mode analog comparators.*
- `bool enableCommonThresholdDetection`  
*Increase the threshold detection for common mode analog comparator.*
- `bool enableInvertHysteresisSign`  
*Invert the sign of the hysteresis in DC-DC analog comparators.*
- `bool enableRCThresholdDetection`  
*Increase the threshold detection for RC scale circuit.*
- `uint32_t enableRCScaleCircuit`  
*Available range is 0~7.*
- `uint32_t complementFeedForwardStep`  
*Available range is 0~7.*

#### Field Documentation

- (1) `bool dcdc_loop_control_config_t::enableCommonHysteresis`

This feature will improve transient supply ripple and efficiency.

- (2) `bool dcdc_loop_control_config_t::enableCommonThresholdDetection`
- (3) `bool dcdc_loop_control_config_t::enableInvertHysteresisSign`
- (4) `bool dcdc_loop_control_config_t::enableRCThresholdDetection`
- (5) `uint32_t dcdc_loop_control_config_t::enableRCScaleCircuit`

Enable analog circuit of DC-DC converter to respond faster under transient load conditions.

- (6) `uint32_t dcdc_loop_control_config_t::complementFeedForwardStep`

Two's complement feed forward step in duty cycle in the switching DC-DC converter. Each time this field makes a transition from 0x0, the loop filter of the DC-DC converter is stepped once by a value proportional to the change. This can be used to force a certain control loop behavior, such as improving response under known heavy load transients.

### 15.4.3 struct `dcdc_low_power_config_t`

#### Data Fields

- `bool enableOverloadDetection`  
*Enable the overload detection in power save mode, if current is larger than the overloading threshold (typical value is 50 mA), DCDC will switch to the run mode automatically.*
- `bool enableAdjustHystereticValue`  
*Adjust hysteretic value in low power from 12.5mV to 25mV.*
- `dcdc_count_charging_time_period_t countChargingTimePeriod`  
*The period of counting the charging times in power save mode.*
- `dcdc_count_charging_time_threshold_t countChargingTimeThreshold`  
*the threshold of the counting number of charging times during the period that `lp_overload_freq_sel` sets in power save mode.*

## Field Documentation

- (1) `bool dcdc_low_power_config_t::enableOverloadDetection`
- (2) `bool dcdc_low_power_config_t::enableAdjustHystereticValue`
- (3) `dcdc_count_charging_time_period_t dcdc_low_power_config_t::countChargingTimePeriod`
- (4) `dcdc_count_charging_time_threshold_t dcdc_low_power_config_t::countChargingTimeThreshold`

15.4.4 `struct dcdc_internal_regulator_config_t`

## Data Fields

- `bool enableLoadResistor`  
*control the load resistor of the internal regulator of DCDC, the load resistor is connected as default "true", and need set to "false" to disconnect the load resistor.*
- `uint32_t feedbackPoint`  
*Available range is 0~3.*

## Field Documentation

- (1) `bool dcdc_internal_regulator_config_t::enableLoadResistor`
- (2) `uint32_t dcdc_internal_regulator_config_t::feedbackPoint`

Select the feedback point of the internal regulator.

15.4.5 `struct dcdc_min_power_config_t`

## Data Fields

- `bool enableUseHalfFreqForContinuous`  
*Set DCDC clock to half frequency for the continuous mode.*

## Field Documentation

- (1) `bool dcdc_min_power_config_t::enableUseHalfFreqForContinuous`

## 15.5 Macro Definition Documentation

15.5.1 `#define FSL_DCDC_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))`

Version 2.3.0.

## 15.6 Enumeration Type Documentation

### 15.6.1 enum \_dcdc\_status\_flags\_t

Enumerator

*kDCDC\_LockedOKStatus* Indicate DCDC status. 1'b1: DCDC already settled 1'b0: DCDC is settling.

### 15.6.2 enum dcdc\_comparator\_current\_bias\_t

Enumerator

*kDCDC\_ComparatorCurrentBias50nA* The current bias of low power comparator is 50nA.  
*kDCDC\_ComparatorCurrentBias100nA* The current bias of low power comparator is 100nA.  
*kDCDC\_ComparatorCurrentBias200nA* The current bias of low power comparator is 200nA.  
*kDCDC\_ComparatorCurrentBias400nA* The current bias of low power comparator is 400nA.

### 15.6.3 enum dcdc\_over\_current\_threshold\_t

Enumerator

*kDCDC\_OverCurrentThresholdAlt0* 1A in the run mode, 0.25A in the power save mode.  
*kDCDC\_OverCurrentThresholdAlt1* 2A in the run mode, 0.25A in the power save mode.  
*kDCDC\_OverCurrentThresholdAlt2* 1A in the run mode, 0.2A in the power save mode.  
*kDCDC\_OverCurrentThresholdAlt3* 2A in the run mode, 0.2A in the power save mode.

### 15.6.4 enum dcdc\_peak\_current\_threshold\_t

Enumerator

*kDCDC\_PeakCurrentThresholdAlt0* 150mA peak current threshold.  
*kDCDC\_PeakCurrentThresholdAlt1* 250mA peak current threshold.  
*kDCDC\_PeakCurrentThresholdAlt2* 350mA peak current threshold.  
*kDCDC\_PeakCurrentThresholdAlt3* 450mA peak current threshold.  
*kDCDC\_PeakCurrentThresholdAlt4* 550mA peak current threshold.  
*kDCDC\_PeakCurrentThresholdAlt5* 650mA peak current threshold.



### 15.6.5 enum dc\_dc\_count\_charging\_time\_period\_t

Enumerator

*kDCDC\_CountChargingTimePeriod8Cycle* Eight 32k cycle.  
*kDCDC\_CountChargingTimePeriod16Cycle* Sixteen 32k cycle.

### 15.6.6 enum dc\_dc\_count\_charging\_time\_threshold\_t

Enumerator

*kDCDC\_CountChargingTimeThreshold32* 0x0: 32.  
*kDCDC\_CountChargingTimeThreshold64* 0x1: 64.  
*kDCDC\_CountChargingTimeThreshold16* 0x2: 16.  
*kDCDC\_CountChargingTimeThreshold8* 0x3: 8.

### 15.6.7 enum dc\_dc\_clock\_source\_t

Enumerator

*kDCDC\_ClockAutoSwitch* Automatic clock switch from internal oscillator to external clock.  
*kDCDC\_ClockInternalOsc* Use internal oscillator.  
*kDCDC\_ClockExternalOsc* Use external 24M crystal oscillator.

## 15.7 Function Documentation

### 15.7.1 void DCDC\_Init ( DCDC\_Type \* *base* )

Parameters

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

### 15.7.2 void DCDC\_Deinit ( DCDC\_Type \* *base* )

Parameters

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

### 15.7.3 uint32\_t DCDC\_GetstatusFlags ( DCDC\_Type \* *base* )

Parameters

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

Returns

Mask of asserted status flags. See to "\_dcdc\_status\_flags\_t".

### 15.7.4 static void DCDC\_EnableOutputRangeComparator ( DCDC\_Type \* *base*, bool *enable* ) [inline], [static]

The output range comparator is disabled by default.

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | DCDC peripheral base address. |
| <i>enable</i> | Enable the feature or not.    |

### 15.7.5 void DCDC\_SetClockSource ( DCDC\_Type \* *base*, dcdc\_clock\_source\_t *clockSource* )

Parameters

|                    |                                                      |
|--------------------|------------------------------------------------------|
| <i>base</i>        | DCDC peripheral base address.                        |
| <i>clockSource</i> | Clock source for DCDC. See to "dcdc_clock_source_t". |

### 15.7.6 void DCDC\_GetDefaultDetectionConfig ( dcdc\_detection\_config\_t \* *config* )

The default configuration are set according to responding registers' setting when powered on. They are:

```
* config->enableXtalokDetection = false;
* config->powerDownOverVoltageDetection = true;
* config->powerDownLowVoltageDetection = false;
```

```
* config->powerDownOverCurrentDetection = true;
* config->powerDownPeakCurrentDetection = true;
* config->powerDownZeroCrossDetection = true;
* config->OverCurrentThreshold = kDCDC_OverCurrentThresholdAlt0;
* config->PeakCurrentThreshold = kDCDC_PeakCurrentThresholdAlt0;
*
```

Parameters

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

**15.7.7 void DCDC\_SetDetectionConfig ( DCDC\_Type \* *base*, const dcdc\_detection\_config\_t \* *config* )**

Parameters

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

**15.7.8 void DCDC\_GetDefaultLowPowerConfig ( dcdc\_low\_power\_config\_t \* *config* )**

The default configuration are set according to responding registers' setting when powered on. They are:

```
* config->enableOverloadDetection = true;
* config->enableAdjustHystereticValue = false;
* config->countChargingTimePeriod = kDCDC_CountChargingTimePeriod8Cycle
;
* config->countChargingTimeThreshold = kDCDC_CountChargingTimeThreshold32
;
*
```

Parameters

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

**15.7.9 void DCDC\_SetLowPowerConfig ( DCDC\_Type \* *base*, const dcdc\_low\_power\_config\_t \* *config* )**

## Parameters

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

**15.7.10 void DCDC\_ResetCurrentAlertSignal ( DCDC\_Type \* *base*, bool *enable* )**

Alert signal is generate by peak current detection.

## Parameters

|               |                                                                                    |
|---------------|------------------------------------------------------------------------------------|
| <i>base</i>   | DCDC peripheral base address.                                                      |
| <i>enable</i> | Switcher to reset signal. True means reset signal. False means don't reset signal. |

**15.7.11 static void DCDC\_SetBandgapVoltageTrimValue ( DCDC\_Type \* *base*, uint32\_t *trimValue* ) [inline], [static]**

## Parameters

|                  |                                                   |
|------------------|---------------------------------------------------|
| <i>base</i>      | DCDC peripheral base address.                     |
| <i>trimValue</i> | The bangap trim value. Available range is 0U-31U. |

**15.7.12 void DCDC\_GetDefaultLoopControlConfig ( dcdc\_loop\_control\_config\_t \* *config* )**

The default configuration are set according to responding registers' setting when powered on. They are:

```
* config->enableCommonHysteresis = false;
* config->enableCommonThresholdDetection = false;
* config->enableInvertHysteresisSign = false;
* config->enableRCThresholdDetection = false;
* config->enableRCScaleCircuit = 0U;
* config->complementFeedForwardStep = 0U;
*
```

Parameters

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

**15.7.13 void DCDC\_SetLoopControlConfig ( DCDC\_Type \* *base*, const dcdc\_loop\_control\_config\_t \* *config* )**

Parameters

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

**15.7.14 void DCDC\_SetMinPowerConfig ( DCDC\_Type \* *base*, const dcdc\_min\_power\_config\_t \* *config* )**

Parameters

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

**15.7.15 static void DCDC\_SetLPComparatorBiasValue ( DCDC\_Type \* *base*, dcdc\_comparator\_current\_bias\_t *biasVaule* ) [inline], [static]**

Parameters

|                  |                                                                                      |
|------------------|--------------------------------------------------------------------------------------|
| <i>base</i>      | DCDC peripheral base address.                                                        |
| <i>biasVaule</i> | The current bias of low power comparator. Refer to "dcdc_comparator_current_bias_t". |

**15.7.16 static void DCDC\_LockTargetVoltage ( DCDC\_Type \* *base* ) [inline], [static]**

## Parameters

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

### 15.7.17 void DCDC\_AdjustTargetVoltage ( DCDC\_Type \* *base*, uint32\_t *VDDRun*, uint32\_t *VDDStandby* )

**Deprecated** Do not use this function. It has been superseded by [DCDC\\_AdjustRunTargetVoltage](#) and [DCDC\\_AdjustLowPowerTargetVoltage](#)

This function is to adjust the target voltage of DCDC output. Change them and finally wait until the output is stabled. Set the target value of run mode the same as low power mode before entering power save mode, because DCDC will switch back to run mode if it detects the current loading is larger than about 50 mA(typical value).

## Parameters

|                   |                                                                                                    |
|-------------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>       | DCDC peripheral base address.                                                                      |
| <i>VDDRun</i>     | Target value in run mode. 25 mV each step from 0x00 to 0x1F. 00 is for 0.8V, 0x1F is for 1.575V.   |
| <i>VDDStandby</i> | Target value in low power mode. 25 mV each step from 0x00 to 0x4. 00 is for 0.9V, 0x4 is for 1.0V. |

### 15.7.18 void DCDC\_AdjustRunTargetVoltage ( DCDC\_Type \* *base*, uint32\_t *VDDRun* )

This function is to adjust the target voltage of DCDC output. Change them and finally wait until the output is stabled. Set the target value of run mode the same as low power mode before entering power save mode, because DCDC will switch back to run mode if it detects the current loading is larger than about 50 mA(typical value).

## Parameters

|               |                                                                                                  |
|---------------|--------------------------------------------------------------------------------------------------|
| <i>base</i>   | DCDC peripheral base address.                                                                    |
| <i>VDDRun</i> | Target value in run mode. 25 mV each step from 0x00 to 0x1F. 00 is for 0.8V, 0x1F is for 1.575V. |

### 15.7.19 void DCDC\_AdjustLowPowerTargetVoltage ( DCDC\_Type \* *base*, uint32\_t *VDDStandby* )

This function is to adjust the target voltage of DCDC output. Change them and finally wait until the output is stabled. Set the target value of run mode the same as low power mode before entering power save mode, because DCDC will switch back to run mode if it detects the current loading is larger than about 50 mA(typical value).

Parameters

|                   |                                                                                                    |
|-------------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>       | DCDC peripheral base address.                                                                      |
| <i>VDDStandby</i> | Target value in low power mode. 25 mV each step from 0x00 to 0x4. 00 is for 0.9V, 0x4 is for 1.0V. |

### 15.7.20 void DCDC\_SetInternalRegulatorConfig ( DCDC\_Type \* *base*, const *dcdc\_internal\_regulator\_config\_t* \* *config* )

Parameters

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

### 15.7.21 static void DCDC\_EnableImproveTransition ( DCDC\_Type \* *base*, bool *enable* ) [inline], [static]

It is valid while zero cross detection is enabled. If output exceeds the threshold, DCDC would return CCM from DCM.

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | DCDC peripheral base address. |
| <i>enable</i> | Enable the feature or not.    |

### 15.7.22 void DCDC\_BootIntoDCM ( DCDC\_Type \* *base* )

`pwd_zcd=0x0; pwd_cmp_offset=0x0; dcdc_loopctrl_en_rcscale= 0x5; DCM_set_ctrl=1'b1;`

Parameters

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

### 15.7.23 void DCDC\_BootIntoCCM ( DCDC\_Type \* *base* )

`pwd_zcd=0x1; pwd_cmp_offset=0x0; dcdc_loopctrl_en_rcscale=0x3;`

Parameters

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



# Chapter 16

## DCP: Data Co-Processor

### 16.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Data Co-Processor (DCP) module. For security purposes, the Data Co-Processor (DCP) provides hardware acceleration for the cryptographic algorithms. The features of DCP are: Encryption Algorithms: AES-128 (ECB and CBC modes), Hashing Algorithms: SHA-1 and SHA-256, modified CRC-32, Key selection from the SNVS, DCP internal key storage, or general memory, Internal Memory for storing up to four AES-128 keys-when a key is written to a key slot it can be read only by the DCP AES-128 engine, IP slave interface, and DMA.

The driver comprises two sets of API functions.

In the first set, blocking APIs are provided, for selected subset of operations supported by DCP hardware. The DCP operations are complete (and results are made available for further usage) when a function returns. When called, these functions do not return until a DCP operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status.

The DCP work packets (descriptors) are placed on the system stack during the blocking API calls. The driver uses critical section (implemented as global interrupt enable/disable) for a short time whenever it needs to pass DCP work packets to DCP channel for processing. Therefore, the driver functions are designed to be re-entrant and as a consequence, one CPU thread can call one blocking API, such as AES Encrypt, while other CPU thread can call another blocking API, such as SHA-256 Update. The blocking functions provide typical interface to upper layer or application software.

In the second set, non-blocking variants of the first set APIs are provided. Internally, the blocking APIs are implemented as a non-blocking operation start, followed by a blocking wait (CPU polling DCP work packet's status word) for an operation completion. The non-blocking functions allow upper layer to inject an application specific operation after the DCP operation start and DCP channel complete events. RTOS event wait and RTOS event set can be an example of such an operation.

### 16.2 DCP Driver Initialization and Configuration

Initialize DCP after Power On Reset or reset cycle Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/dcp

The DCP Driver is initialized by calling the [DCP\\_Init\(\)](#) function. It enables the DCP module clock and configures DCP for operation.

### Key Management

The DCP implements four different key storage mechanisms: OTP-key, OTP-Unique key, Payload key, and SRAM-based keys that can be used by the software to securely store keys on a semi-permanent basis (kDCP\_KeySlot0 ~ kDCP\_KeySlot3). Once the function [DCP\\_AES\\_SetKey\(\)](#) is called, it sets the AES key for encryption/decryption with the [dcp\\_handle\\_t](#) structure. In case the SRAM-based key is selected,

the function copies and holds the key in memory. In case the OTP key is used, please make sure to set DCP related IOMUXC\_GPRs before DCP initialization, since the software reset of DCP must be issued to take these setting in effect. Refer to the DCP\_OTPKeySelect() function in BEE driver example.

### 16.3 Comments about API usage in RTOS

DCP transactional (encryption or hash) APIs can be called from multiple threads.

### 16.4 Comments about API usage in interrupt handler

Assuming the host processor receiving interrupt has the ownership of the DCP module, it can request Encrypt/Decrypt/Hash/public\_key operations in an interrupt routine. Additionally, as the DCP accesses system memory for it's operation with data (such as message, plaintext, ciphertext, or keys) all data should remain valid until the DCP operation completes.

### 16.5 Comments about DCACHE

Input and output buffers passed to DCP API should be in non-cached memory or handled properly (DCACHE Clean and Invalidate) while using DCACHE.

## 16.6 DCP Driver Examples

### 16.6.1 Simple examples

Encrypt plaintext by AES engine Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/DCP

Compute hash (CRC-32) The CRC-32 algorithm implements a 32-bit CRC algorithm similar to the one used by Ethernet and many other protocols. The CRC differs from the Unix cksum() function in these four ways: The CRC initial value is 0xFFFFFFFF instead of 0x00000000, final XOR value is 0x00000000 instead of 0xFFFFFFFF, the logic pads the zeros to a 32-bit boundary for the trailing bytes, and it does not post-pend the file length. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/DCP

Compute hash (SHA-256) Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/DCP

## Modules

- [DCP AES blocking driver](#)
- [DCP AES non-blocking driver](#)
- [DCP HASH driver](#)

## Data Structures

- struct [dcp\\_work\\_packet\\_t](#)  
*DCP's work packet. [More...](#)*
- struct [dcp\\_handle\\_t](#)  
*Specify DCP's key resource and DCP channel. [More...](#)*

- struct `dcp_context_t`  
*DCP's context buffer, used by DCP for context switching between channels. [More...](#)*
- struct `dcp_config_t`  
*DCP's configuration structure. [More...](#)*

## Enumerations

- enum `_dcp_status` { `kStatus_DCP_Again` = MAKE\_STATUS(kStatusGroup\_DCP, 0) }  
*DCP status return codes.*
- enum `_dcp_ch_enable_t` {  
`kDCP_chDisable` = 0U,  
`kDCP_ch0Enable` = 1U,  
`kDCP_ch1Enable` = 2U,  
`kDCP_ch2Enable` = 4U,  
`kDCP_ch3Enable` = 8U,  
`kDCP_chEnableAll` = 15U }  
*DCP channel enable.*
- enum `_dcp_ch_int_enable_t` {  
`kDCP_chIntDisable` = 0U,  
`kDCP_ch0IntEnable` = 1U,  
`kDCP_ch1IntEnable` = 2U,  
`kDCP_ch2IntEnable` = 4U,  
`kDCP_ch3IntEnable` = 8U }  
*DCP interrupt enable.*
- enum `dcp_channel_t` {  
`kDCP_Channel0` = (1u << 16),  
`kDCP_Channel1` = (1u << 17),  
`kDCP_Channel2` = (1u << 18),  
`kDCP_Channel3` = (1u << 19) }  
*DCP channel selection.*
- enum `dcp_key_slot_t` {  
`kDCP_KeySlot0` = 0U,  
`kDCP_KeySlot1` = 1U,  
`kDCP_KeySlot2` = 2U,  
`kDCP_KeySlot3` = 3U,  
`kDCP_OtpKey` = 4U,  
`kDCP_OtpUniqueKey` = 5U,  
`kDCP_PayloadKey` = 6U }  
*DCP key slot selection.*
- enum `dcp_swap_t`  
*DCP key, input & output swap options.*

## Functions

- void `DCP_Init` (DCP\_Type \*base, const `dcp_config_t` \*config)  
*Enables clock to and enables DCP.*
- void `DCP_Deinit` (DCP\_Type \*base)  
*Disable DCP clock.*

- void [DCP\\_GetDefaultConfig](#) ([dcp\\_config\\_t](#) \*config)  
*Gets the default configuration structure.*
- [status\\_t DCP\\_WaitForChannelComplete](#) ([DCP\\_Type](#) \*base, [dcp\\_handle\\_t](#) \*handle)  
*Poll and wait on DCP channel.*

## Driver version

- #define [FSL\\_DCP\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 7))  
*DCP driver version.*

## 16.7 Data Structure Documentation

### 16.7.1 struct [dcp\\_work\\_packet\\_t](#)

### 16.7.2 struct [dcp\\_handle\\_t](#)

#### Data Fields

- [dcp\\_channel\\_t](#) channel  
*Specify DCP channel.*
- [dcp\\_key\\_slot\\_t](#) keySlot  
*For operations with key (such as AES encryption/decryption), specify DCP key slot.*
- [uint32\\_t](#) [swapConfig](#)  
*For configuration of key, input, output byte/word swap options.*

#### Field Documentation

(1) [dcp\\_channel\\_t](#) [dcp\\_handle\\_t::channel](#)

(2) [dcp\\_key\\_slot\\_t](#) [dcp\\_handle\\_t::keySlot](#)

### 16.7.3 struct [dcp\\_context\\_t](#)

### 16.7.4 struct [dcp\\_config\\_t](#)

#### Data Fields

- bool [gatherResidualWrites](#)  
*Enable the ragged writes to the unaligned buffers.*
- bool [enableContextCaching](#)  
*Enable the caching of contexts between the operations.*
- bool [enableContextSwitching](#)  
*Enable automatic context switching for the channels.*
- [uint8\\_t](#) [enableChannel](#)  
*DCP channel enable.*
- [uint8\\_t](#) [enableChannelInterrupt](#)  
*Per-channel interrupt enable.*

## Field Documentation

- (1) `bool dcp_config_t::gatherResidualWrites`
- (2) `bool dcp_config_t::enableContextCaching`
- (3) `bool dcp_config_t::enableContextSwitching`
- (4) `uint8_t dcp_config_t::enableChannel`
- (5) `uint8_t dcp_config_t::enableChannelInterrupt`

## 16.8 Macro Definition Documentation

### 16.8.1 `#define FSL_DCP_DRIVER_VERSION (MAKE_VERSION(2, 1, 7))`

Version 2.1.7.

Current version: 2.1.7

Change log:

- Version 2.1.7
  - Bug Fix
    - \* Reduce optimization level for critical functions working with SRF.
- Version 2.1.6
  - Bug Fix
    - \* MISRA C-2012 issue fix.
- Version 2.1.5
  - Improvements
    - \* Add support for DCACHE.
- Version 2.1.4
  - Bug Fix
    - \* Fix CRC-32 computation issue on the code's block boundary size.
- Version 2.1.3
  - Bug Fix
    - \* MISRA C-2012 issue fixed: rule 10.1, 10.3, 10.4, 11.9, 14.4, 16.4 and 17.7.
- Version 2.1.2
  - Fix sign-compare warning in `dcp_reverse_and_copy`.
- Version 2.1.1
  - Add DCP status clearing when channel operation is complete
- 2.1.0
  - Add byte/word swap feature for key, input and output data
- Version 2.0.0
  - Initial version

## 16.9 Enumeration Type Documentation

### 16.9.1 enum \_dcp\_status

Enumerator

*kStatus\_DCP\_Again* Non-blocking function shall be called again.

### 16.9.2 enum \_dcp\_ch\_enable\_t

Enumerator

*kDCP\_chDisable* DCP channel disable.  
*kDCP\_ch0Enable* DCP channel 0 enable.  
*kDCP\_ch1Enable* DCP channel 1 enable.  
*kDCP\_ch2Enable* DCP channel 2 enable.  
*kDCP\_ch3Enable* DCP channel 3 enable.  
*kDCP\_chEnableAll* DCP channel enable all.

### 16.9.3 enum \_dcp\_ch\_int\_enable\_t

Enumerator

*kDCP\_chIntDisable* DCP interrupts disable.  
*kDCP\_ch0IntEnable* DCP channel 0 interrupt enable.  
*kDCP\_ch1IntEnable* DCP channel 1 interrupt enable.  
*kDCP\_ch2IntEnable* DCP channel 2 interrupt enable.  
*kDCP\_ch3IntEnable* DCP channel 3 interrupt enable.

### 16.9.4 enum dcp\_channel\_t

Enumerator

*kDCP\_Channel0* DCP channel 0.  
*kDCP\_Channel1* DCP channel 1.  
*kDCP\_Channel2* DCP channel 2.  
*kDCP\_Channel3* DCP channel 3.

### 16.9.5 enum dcp\_key\_slot\_t

Enumerator

***kDCP\_KeySlot0*** DCP key slot 0.  
***kDCP\_KeySlot1*** DCP key slot 1.  
***kDCP\_KeySlot2*** DCP key slot 2.  
***kDCP\_KeySlot3*** DCP key slot 3.  
***kDCP\_OtpKey*** DCP OTP key.  
***kDCP\_OtpUniqueKey*** DCP unique OTP key.  
***kDCP\_PayloadKey*** DCP payload key.

### 16.9.6 enum dcp\_swap\_t

## 16.10 Function Documentation

### 16.10.1 void DCP\_Init ( DCP\_Type \* *base*, const dcp\_config\_t \* *config* )

Enable DCP clock and configure DCP.

Parameters

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

### 16.10.2 void DCP\_Deinit ( DCP\_Type \* *base* )

Reset DCP and Disable DCP clock.

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | DCP base address |
|-------------|------------------|

### 16.10.3 void DCP\_GetDefaultConfig ( dcp\_config\_t \* *config* )

This function initializes the DCP configuration structure to a default value. The default values are as follows. `dcpConfig->gatherResidualWrites = true;` `dcpConfig->enableContextCaching = true;` `dcpConfig->enableContextSwitching = true;` `dcpConfig->enableChannel = kDCP_chEnableAll;` `dcpConfig->enableChannelInterrupt = kDCP_chIntDisable;`

Parameters

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

#### 16.10.4 status\_t DCP\_WaitForChannelComplete ( DCP\_Type \* *base*, dcp\_handle\_t \* *handle* )

Polls the specified DCP channel until current it completes activity.

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | DCP peripheral base address. |
| <i>handle</i> | Specifies DCP channel.       |

Returns

kStatus\_Success When data processing completes without error.

kStatus\_Fail When error occurs.



## 16.11 DCP AES blocking driver

### 16.11.1 Overview

This section describes the programming interface of the DCP AES blocking driver.

#### Macros

- #define `DCP_AES_BLOCK_SIZE` 16  
*AES block size in bytes.*

#### Functions

- `status_t DCP_AES_SetKey` (DCP\_Type \*base, `dcp_handle_t` \*handle, const uint8\_t \*key, size\_t keySize)  
*Set AES key to `dcp_handle_t` struct and optionally to DCP.*
- `status_t DCP_AES_EncryptEcb` (DCP\_Type \*base, `dcp_handle_t` \*handle, const uint8\_t \*plaintext, uint8\_t \*ciphertext, size\_t size)  
*Encrypts AES on one or multiple 128-bit block(s).*
- `status_t DCP_AES_DecryptEcb` (DCP\_Type \*base, `dcp_handle_t` \*handle, const uint8\_t \*ciphertext, uint8\_t \*plaintext, size\_t size)  
*Decrypts AES on one or multiple 128-bit block(s).*
- `status_t DCP_AES_EncryptCbc` (DCP\_Type \*base, `dcp_handle_t` \*handle, const uint8\_t \*plaintext, uint8\_t \*ciphertext, size\_t size, const uint8\_t iv[16])  
*Encrypts AES using CBC block mode.*
- `status_t DCP_AES_DecryptCbc` (DCP\_Type \*base, `dcp_handle_t` \*handle, const uint8\_t \*ciphertext, uint8\_t \*plaintext, size\_t size, const uint8\_t iv[16])  
*Decrypts AES using CBC block mode.*

### 16.11.2 Function Documentation

#### 16.11.2.1 `status_t DCP_AES_SetKey ( DCP_Type * base, dcp_handle_t * handle, const uint8_t * key, size_t keySize )`

Sets the AES key for encryption/decryption with the `dcp_handle_t` structure. The `dcp_handle_t` input argument specifies keySlot. If the keySlot is `kDCP_OtpKey`, the function will check the `OTP_KEY_READY` bit and will return it's ready to use status. For other keySlot selections, the function will copy and hold the key in `dcp_handle_t` struct. If the keySlot is one of the four DCP SRAM-based keys (one of `kDCP_KeySlot0`, `kDCP_KeySlot1`, `kDCP_KeySlot2`, `kDCP_KeySlot3`), this function will also load the supplied key to the specified keySlot in DCP.

## Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | DCP peripheral base address.           |
| <i>handle</i>  | Handle used for the request.           |
| <i>key</i>     | 0-mod-4 aligned pointer to AES key.    |
| <i>keySize</i> | AES key size in bytes. Shall equal 16. |

## Returns

status from set key operation

**16.11.2.2** `status_t DCP_AES_EncryptEcb ( DCP_Type * base, dcp_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size )`

Encrypts AES. The source plaintext and destination ciphertext can overlap in system memory.

## Parameters

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

## Returns

Status from encrypt operation

**16.11.2.3** `status_t DCP_AES_DecryptEcb ( DCP_Type * base, dcp_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size )`

Decrypts AES. The source ciphertext and destination plaintext can overlap in system memory.

## Parameters

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

## Returns

Status from decrypt operation

**16.11.2.4** `status_t DCP_AES_EncryptCbc ( DCP_Type * base, dcp_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[16] )`

Encrypts AES using CBC block mode. The source plaintext and destination ciphertext can overlap in system memory.

## Parameters

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

## Returns

Status from encrypt operation

**16.11.2.5** `status_t DCP_AES_DecryptCbc ( DCP_Type * base, dcp_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[16] )`

Decrypts AES using CBC block mode. The source ciphertext and destination plaintext can overlap in system memory.

## Parameters

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

## Returns

Status from decrypt operation

## 16.12 DCP AES non-blocking driver

### 16.12.1 Overview

This section describes the programming interface of the DCP AES non-blocking driver.

### Functions

- `status_t DCP_AES_EncryptEcbNonBlocking` (DCP\_Type \*base, `dcp_handle_t` \*handle, `dcp_work_packet_t` \*dcpPacket, const uint8\_t \*plaintext, uint8\_t \*ciphertext, size\_t size)  
*Encrypts AES using the ECB block mode.*
- `status_t DCP_AES_DecryptEcbNonBlocking` (DCP\_Type \*base, `dcp_handle_t` \*handle, `dcp_work_packet_t` \*dcpPacket, const uint8\_t \*ciphertext, uint8\_t \*plaintext, size\_t size)  
*Decrypts AES using ECB block mode.*
- `status_t DCP_AES_EncryptCbcNonBlocking` (DCP\_Type \*base, `dcp_handle_t` \*handle, `dcp_work_packet_t` \*dcpPacket, const uint8\_t \*plaintext, uint8\_t \*ciphertext, size\_t size, const uint8\_t \*iv)  
*Encrypts AES using CBC block mode.*
- `status_t DCP_AES_DecryptCbcNonBlocking` (DCP\_Type \*base, `dcp_handle_t` \*handle, `dcp_work_packet_t` \*dcpPacket, const uint8\_t \*ciphertext, uint8\_t \*plaintext, size\_t size, const uint8\_t \*iv)  
*Decrypts AES using CBC block mode.*

### 16.12.2 Function Documentation

#### 16.12.2.1 `status_t DCP_AES_EncryptEcbNonBlocking` ( DCP\_Type \* *base*, `dcp_handle_t` \* *handle*, `dcp_work_packet_t` \* *dcpPacket*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, size\_t *size* )

Puts AES ECB encrypt work packet to DCP channel.

Parameters

|     |                   |                                                                       |
|-----|-------------------|-----------------------------------------------------------------------|
|     | <i>base</i>       | DCP peripheral base address                                           |
|     | <i>handle</i>     | Handle used for this request.                                         |
| out | <i>dcpPacket</i>  | Memory for the DCP work packet.                                       |
|     | <i>plaintext</i>  | Input plain text to encrypt.                                          |
| out | <i>ciphertext</i> | Output cipher text                                                    |
|     | <i>size</i>       | Size of input and output data in bytes. Must be multiple of 16 bytes. |

Returns

`kStatus_Success` The work packet has been scheduled at DCP channel.

`kStatus_DCP_Again` The DCP channel is busy processing previous request.

**16.12.2.2** `status_t DCP_AES_DecryptEcbNonBlocking ( DCP_Type * base, dcp_handle_t * handle, dcp_work_packet_t * dcpPacket, const uint8_t * ciphertext, uint8_t * plaintext, size_t size )`

Puts AES ECB decrypt dcpPacket to DCP input job ring.

## Parameters

|     |                   |                                                                       |
|-----|-------------------|-----------------------------------------------------------------------|
|     | <i>base</i>       | DCP peripheral base address                                           |
|     | <i>handle</i>     | Handle used for this request.                                         |
| out | <i>dcpPacket</i>  | Memory for the DCP work packet.                                       |
|     | <i>ciphertext</i> | Input cipher text to decrypt                                          |
| out | <i>plaintext</i>  | Output plain text                                                     |
|     | <i>size</i>       | Size of input and output data in bytes. Must be multiple of 16 bytes. |

## Returns

kStatus\_Success The work packet has been scheduled at DCP channel.

kStatus\_DCP\_Again The DCP channel is busy processing previous request.

**16.12.2.3** `status_t DCP_AES_EncryptCbcNonBlocking ( DCP_Type * base, dcp_handle_t * handle, dcp_work_packet_t * dcpPacket, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t * iv )`

Puts AES CBC encrypt dcpPacket to DCP input job ring.

## Parameters

|     |                   |                                                                       |
|-----|-------------------|-----------------------------------------------------------------------|
|     | <i>base</i>       | DCP peripheral base address                                           |
|     | <i>handle</i>     | Handle used for this request. Specifies jobRing.                      |
| out | <i>dcpPacket</i>  | Memory for the DCP work packet.                                       |
|     | <i>plaintext</i>  | Input plain text to encrypt                                           |
| out | <i>ciphertext</i> | Output cipher text                                                    |
|     | <i>size</i>       | Size of input and output data in bytes. Must be multiple of 16 bytes. |
|     | <i>iv</i>         | Input initial vector to combine with the first input block.           |

## Returns

kStatus\_Success The work packet has been scheduled at DCP channel.

kStatus\_DCP\_Again The DCP channel is busy processing previous request.

**16.12.2.4** `status_t DCP_AES_DecryptCbcNonBlocking ( DCP_Type * base, dcp_handle_t * handle, dcp_work_packet_t * dcpPacket, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t * iv )`

Puts AES CBC decrypt dcpPacket to DCP input job ring.

## Parameters

|     |                   |                                                                       |
|-----|-------------------|-----------------------------------------------------------------------|
|     | <i>base</i>       | DCP peripheral base address                                           |
|     | <i>handle</i>     | Handle used for this request. Specifies jobRing.                      |
| out | <i>dcpPacket</i>  | Memory for the DCP work packet.                                       |
|     | <i>ciphertext</i> | Input cipher text to decrypt                                          |
| out | <i>plaintext</i>  | Output plain text                                                     |
|     | <i>size</i>       | Size of input and output data in bytes. Must be multiple of 16 bytes. |
|     | <i>iv</i>         | Input initial vector to combine with the first input block.           |

## Returns

kStatus\_Success The work packet has been scheduled at DCP channel.

kStatus\_DCP\_Again The DCP channel is busy processing previous request.



## 16.13 DCP HASH driver

### 16.13.1 Overview

This section describes the programming interface of the DCP HASH driver.

#### Data Structures

- struct `dcp_hash_ctx_t`  
*Storage type used to save hash context. [More...](#)*

#### Macros

- #define `DCP_SHA_BLOCK_SIZE` 128U  
*DCP HASH Context size.*
- #define `DCP_HASH_BLOCK_SIZE` `DCP_SHA_BLOCK_SIZE`  
*DCP hash block size.*
- #define `DCP_HASH_CTX_SIZE` 64  
*DCP HASH Context size.*

#### Enumerations

- enum `dcp_hash_algo_t` {  
  `kDCP_Sha1`,  
  `kDCP_Sha256`,  
  `kDCP_Crc32` }  
*Supported cryptographic block cipher functions for HASH creation.*

#### Functions

- `status_t DCP_HASH_Init` (`DCP_Type *base`, `dcp_handle_t *handle`, `dcp_hash_ctx_t *ctx`, `dcp_hash_algo_t algo`)  
*Initialize HASH context.*
- `status_t DCP_HASH_Update` (`DCP_Type *base`, `dcp_hash_ctx_t *ctx`, `const uint8_t *input`, `size_t inputSize`)  
*Add data to current HASH.*
- `status_t DCP_HASH_Finish` (`DCP_Type *base`, `dcp_hash_ctx_t *ctx`, `uint8_t *output`, `size_t *outputSize`)  
*Finalize hashing.*
- `status_t DCP_HASH` (`DCP_Type *base`, `dcp_handle_t *handle`, `dcp_hash_algo_t algo`, `const uint8_t *input`, `size_t inputSize`, `uint8_t *output`, `size_t *outputSize`)  
*Create HASH on given data.*

## 16.13.2 Data Structure Documentation

### 16.13.2.1 struct dcp\_hash\_ctx\_t

## 16.13.3 Macro Definition Documentation

### 16.13.3.1 #define DCP\_SHA\_BLOCK\_SIZE 128U

internal buffer block size

### 16.13.3.2 #define DCP\_HASH\_CTX\_SIZE 64

## 16.13.4 Enumeration Type Documentation

### 16.13.4.1 enum dcp\_hash\_algo\_t

Enumerator

*kDCP\_Sha1* SHA\_1.  
*kDCP\_Sha256* SHA\_256.  
*kDCP\_Crc32* CRC\_32.

## 16.13.5 Function Documentation

### 16.13.5.1 status\_t DCP\_HASH\_Init ( DCP\_Type \* *base*, dcp\_handle\_t \* *handle*, dcp\_hash\_ctx\_t \* *ctx*, dcp\_hash\_algo\_t *algo* )

This function initializes the HASH.

Parameters

|     |               |                                                   |
|-----|---------------|---------------------------------------------------|
|     | <i>base</i>   | DCP peripheral base address                       |
|     | <i>handle</i> | Specifies the DCP channel used for hashing.       |
| out | <i>ctx</i>    | Output hash context                               |
|     | <i>algo</i>   | Underlying algorithm to use for hash computation. |

Returns

Status of initialization

### 16.13.5.2 `status_t DCP_HASH_Update ( DCP_Type * base, dcp_hash_ctx_t * ctx, const uint8_t * input, size_t inputSize )`

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed. The functions blocks. If it returns `kStatus_Success`, the running hash has been updated (DCP has processed the input data), so the memory at the input pointer can be released back to system. The DCP context buffer is updated with the running hash and with all necessary information to support possible context switch.

Parameters

|                |                  |                             |
|----------------|------------------|-----------------------------|
|                | <i>base</i>      | DCP peripheral base address |
| <i>in, out</i> | <i>ctx</i>       | HASH context                |
|                | <i>input</i>     | Input data                  |
|                | <i>inputSize</i> | Size of input data in bytes |

Returns

Status of the hash update operation

### 16.13.5.3 `status_t DCP_HASH_Finish ( DCP_Type * base, dcp_hash_ctx_t * ctx, uint8_t * output, size_t * outputSize )`

Outputs the final hash (computed by [DCP\\_HASH\\_Update\(\)](#)) and erases the context.

Parameters

|                |                   |                                                                                                                                                                                         |
|----------------|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                | <i>base</i>       | DCP peripheral base address                                                                                                                                                             |
| <i>in, out</i> | <i>ctx</i>        | Input hash context                                                                                                                                                                      |
| <i>out</i>     | <i>output</i>     | Output hash data                                                                                                                                                                        |
| <i>in, out</i> | <i>outputSize</i> | Optional parameter (can be passed as NULL). On function entry, it specifies the size of <code>output[]</code> buffer. On function return, it stores the number of updated output bytes. |

Returns

Status of the hash finish operation

### 16.13.5.4 `status_t DCP_HASH ( DCP_Type * base, dcp_handle_t * handle, dcp_hash_algo_t algo, const uint8_t * input, size_t inputSize, uint8_t * output, size_t * outputSize )`

Perform the full SHA or CRC32 in one function call. The function is blocking.

## Parameters

|     |                   |                                                               |
|-----|-------------------|---------------------------------------------------------------|
|     | <i>base</i>       | DCP peripheral base address                                   |
|     | <i>handle</i>     | Handle used for the request.                                  |
|     | <i>algo</i>       | Underlying algorithm to use for hash computation.             |
|     | <i>input</i>      | Input data                                                    |
|     | <i>inputSize</i>  | Size of input data in bytes                                   |
| out | <i>output</i>     | Output hash data                                              |
| out | <i>outputSize</i> | Output parameter storing the size of the output hash in bytes |

## Returns

Status of the one call hash operation.

## Chapter 17

# DMAMUX: Direct Memory Access Multiplexer Driver

### 17.1 Overview

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

### 17.2 Typical use case

#### 17.2.1 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, 1, 0))`  
*DMAMUX driver version 2.1.0.*

#### 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, int32\_t source)  
*Configures the DMAMUX channel source.*

### 17.3 Macro Definition Documentation

#### 17.3.1 `#define FSL_DMAMUX_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

### 17.4 Function Documentation

#### 17.4.1 void `DMAMUX_Init` ( DMAMUX\_Type \* *base* )

This function ungates the DMAMUX clock.

Parameters

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

#### 17.4.2 void DMAMUX\_Deinit ( DMAMUX\_Type \* *base* )

This function gates the DMAMUX clock.

Parameters

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

#### 17.4.3 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.          |

#### 17.4.4 static void DMAMUX\_DisableChannel ( DMAMUX\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This function disables the DMAMUX channel.

Note

The user must disable the DMAMUX channel before configuring it.

Parameters

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

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

**17.4.5 static void DMAMUX\_SetSource ( DMAMUX\_Type \* *base*, uint32\_t *channel*, int32\_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. User need to use the dma_request_source_t type as the input parameter. |

## Chapter 18

# eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

### 18.1 Overview

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

### 18.2 Typical use case

#### 18.2.1 eDMA Operation

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

### Data Structures

- struct `edma_config_t`  
*eDMA global configuration structure. [More...](#)*
- struct `edma_transfer_config_t`  
*eDMA transfer configuration [More...](#)*
- struct `edma_channel_Preemption_config_t`  
*eDMA channel priority configuration [More...](#)*
- struct `edma_minor_offset_config_t`  
*eDMA minor offset configuration [More...](#)*
- struct `edma_tcd_t`  
*eDMA TCD. [More...](#)*
- struct `edma_handle_t`  
*eDMA transfer handle structure [More...](#)*

### Macros

- #define `DMA_DCHPRI_INDEX(channel)` (((channel) & ~0x03U) | (3U - ((channel)&0x03U)))  
*Compute the offset unit from DCHPRI3.*

### Typedefs

- typedef void(\* `edma_callback` )(struct `_edma_handle` \*handle, void \*userData, bool transferDone, uint32\_t tcds)  
*Define callback function for eDMA.*



## Enumerations

- enum `edma_transfer_size_t` {
  - `kEDMA_TransferSize1Bytes` = 0x0U,
  - `kEDMA_TransferSize2Bytes` = 0x1U,
  - `kEDMA_TransferSize4Bytes` = 0x2U,
  - `kEDMA_TransferSize8Bytes` = 0x3U,
  - `kEDMA_TransferSize16Bytes` = 0x4U,
  - `kEDMA_TransferSize32Bytes` = 0x5U }

*eDMA transfer configuration*
- enum `edma_modulo_t` {
  - `kEDMA_ModuloDisable` = 0x0U,
  - `kEDMA_Modulo2bytes`,
  - `kEDMA_Modulo4bytes`,
  - `kEDMA_Modulo8bytes`,
  - `kEDMA_Modulo16bytes`,
  - `kEDMA_Modulo32bytes`,
  - `kEDMA_Modulo64bytes`,
  - `kEDMA_Modulo128bytes`,
  - `kEDMA_Modulo256bytes`,
  - `kEDMA_Modulo512bytes`,
  - `kEDMA_Modulo1Kbytes`,
  - `kEDMA_Modulo2Kbytes`,
  - `kEDMA_Modulo4Kbytes`,
  - `kEDMA_Modulo8Kbytes`,
  - `kEDMA_Modulo16Kbytes`,
  - `kEDMA_Modulo32Kbytes`,
  - `kEDMA_Modulo64Kbytes`,
  - `kEDMA_Modulo128Kbytes`,
  - `kEDMA_Modulo256Kbytes`,
  - `kEDMA_Modulo512Kbytes`,
  - `kEDMA_Modulo1Mbytes`,
  - `kEDMA_Modulo2Mbytes`,
  - `kEDMA_Modulo4Mbytes`,
  - `kEDMA_Modulo8Mbytes`,
  - `kEDMA_Modulo16Mbytes`,
  - `kEDMA_Modulo32Mbytes`,
  - `kEDMA_Modulo64Mbytes`,
  - `kEDMA_Modulo128Mbytes`,
  - `kEDMA_Modulo256Mbytes`,
  - `kEDMA_Modulo512Mbytes`,
  - `kEDMA_Modulo1Gbytes`,
  - `kEDMA_Modulo2Gbytes` }

*eDMA modulo configuration*
- enum `edma_bandwidth_t` {

```
kEDMA_BandwidthStallNone = 0x0U,
kEDMA_BandwidthStall4Cycle = 0x2U,
kEDMA_BandwidthStall8Cycle = 0x3U }
```

*Bandwidth control.*

- enum `edma_channel_link_type_t` {
 

```
kEDMA_LinkNone = 0x0U,
kEDMA_MinorLink,
kEDMA_MajorLink }
```

*Channel link type.*

- enum {
 

```
kEDMA_DoneFlag = 0x1U,
kEDMA_ErrorFlag = 0x2U,
kEDMA_InterruptFlag = 0x4U }
```

*\_edma\_channel\_status\_flags eDMA channel status flags.*

- enum {
 

```
kEDMA_DestinationBusErrorFlag = DMA_ES_DBE_MASK,
kEDMA_SourceBusErrorFlag = DMA_ES_SBE_MASK,
kEDMA_ScatterGatherErrorFlag = DMA_ES_SGE_MASK,
kEDMA_NbytesErrorFlag = DMA_ES_NCE_MASK,
kEDMA_DestinationOffsetErrorFlag = DMA_ES_DOE_MASK,
kEDMA_DestinationAddressErrorFlag = DMA_ES_DAE_MASK,
kEDMA_SourceOffsetErrorFlag = DMA_ES_SOE_MASK,
kEDMA_SourceAddressErrorFlag = DMA_ES_SAE_MASK,
kEDMA_ErrorChannelFlag = DMA_ES_ERRCHN_MASK,
kEDMA_ChannelPriorityErrorFlag = DMA_ES_CPE_MASK,
kEDMA_TransferCanceledFlag = DMA_ES_ECX_MASK,
kEDMA_ValidFlag = (int)DMA_ES_VLD_MASK }
```

*\_edma\_error\_status\_flags eDMA channel error status flags.*

- enum `edma_interrupt_enable_t` {
 

```
kEDMA_ErrorInterruptEnable = 0x1U,
kEDMA_MajorInterruptEnable = DMA_CSR_INTMAJOR_MASK,
kEDMA_HalfInterruptEnable = DMA_CSR_INTHALF_MASK }
```

*eDMA interrupt source*

- enum `edma_transfer_type_t` {
 

```
kEDMA_MemoryToMemory = 0x0U,
kEDMA_PeripheralToMemory,
kEDMA_MemoryToPeripheral,
kEDMA_PeripheralToPeripheral }
```

*eDMA transfer type*

- enum {
 

```
kStatus_EDMA_QueueFull = MAKE_STATUS(kStatusGroup_EDMA, 0),
kStatus_EDMA_Busy = MAKE_STATUS(kStatusGroup_EDMA, 1) }
```

*\_edma\_transfer\_status eDMA transfer status*

## Driver version

- #define `FSL_EDMA_DRIVER_VERSION` (`MAKE_VERSION`(2, 4, 3))

*eDMA driver version*

## eDMA initialization and de-initialization

- void [EDMA\\_Init](#) (DMA\_Type \*base, const [edma\\_config\\_t](#) \*config)  
*Initializes the eDMA peripheral.*
- void [EDMA\\_Deinit](#) (DMA\_Type \*base)  
*Deinitializes the eDMA peripheral.*
- void [EDMA\\_InstallTCD](#) (DMA\_Type \*base, uint32\_t channel, [edma\\_tcd\\_t](#) \*tcd)  
*Push content of TCD structure into hardware TCD register.*
- void [EDMA\\_GetDefaultConfig](#) ([edma\\_config\\_t](#) \*config)  
*Gets the eDMA default configuration structure.*
- static void [EDMA\\_EnableContinuousChannelLinkMode](#) (DMA\_Type \*base, bool enable)  
*Enable/Disable continuous channel link mode.*
- static void [EDMA\\_EnableMinorLoopMapping](#) (DMA\_Type \*base, bool enable)  
*Enable/Disable minor loop mapping.*

## eDMA Channel Operation

- void [EDMA\\_ResetChannel](#) (DMA\_Type \*base, uint32\_t channel)  
*Sets all TCD registers to default values.*
- void [EDMA\\_SetTransferConfig](#) (DMA\_Type \*base, uint32\_t channel, const [edma\\_transfer\\_config\\_t](#) \*config, [edma\\_tcd\\_t](#) \*nextTcd)  
*Configures the eDMA transfer attribute.*
- void [EDMA\\_SetMinorOffsetConfig](#) (DMA\_Type \*base, uint32\_t channel, const [edma\\_minor\\_offset\\_config\\_t](#) \*config)  
*Configures the eDMA minor offset feature.*
- void [EDMA\\_SetChannelPreemptionConfig](#) (DMA\_Type \*base, uint32\_t channel, const [edma\\_channel\\_preemption\\_config\\_t](#) \*config)  
*Configures the eDMA channel preemption feature.*
- void [EDMA\\_SetChannelLink](#) (DMA\_Type \*base, uint32\_t channel, [edma\\_channel\\_link\\_type\\_t](#) linkType, uint32\_t linkedChannel)  
*Sets the channel link for the eDMA transfer.*
- void [EDMA\\_SetBandWidth](#) (DMA\_Type \*base, uint32\_t channel, [edma\\_bandwidth\\_t](#) bandWidth)  
*Sets the bandwidth for the eDMA transfer.*
- void [EDMA\\_SetModulo](#) (DMA\_Type \*base, uint32\_t channel, [edma\\_modulo\\_t](#) srcModulo, [edma\\_modulo\\_t](#) destModulo)  
*Sets the source modulo and the destination modulo for the eDMA transfer.*
- static void [EDMA\\_EnableAsyncRequest](#) (DMA\_Type \*base, uint32\_t channel, bool enable)  
*Enables an async request for the eDMA transfer.*
- static void [EDMA\\_EnableAutoStopRequest](#) (DMA\_Type \*base, uint32\_t channel, bool enable)  
*Enables an auto stop request for the eDMA transfer.*
- void [EDMA\\_EnableChannelInterrupts](#) (DMA\_Type \*base, uint32\_t channel, uint32\_t mask)  
*Enables the interrupt source for the eDMA transfer.*
- void [EDMA\\_DisableChannelInterrupts](#) (DMA\_Type \*base, uint32\_t channel, uint32\_t mask)  
*Disables the interrupt source for the eDMA transfer.*
- void [EDMA\\_SetMajorOffsetConfig](#) (DMA\_Type \*base, uint32\_t channel, int32\_t sourceOffset, int32\_t destOffset)  
*Configures the eDMA channel TCD major offset feature.*

## eDMA TCD Operation

- void [EDMA\\_TcdReset](#) ([edma\\_tcd\\_t](#) \*tcd)  
*Sets all fields to default values for the TCD structure.*
- void [EDMA\\_TcdSetTransferConfig](#) ([edma\\_tcd\\_t](#) \*tcd, const [edma\\_transfer\\_config\\_t](#) \*config, [edma\\_tcd\\_t](#) \*nextTcd)  
*Configures the eDMA TCD transfer attribute.*
- void [EDMA\\_TcdSetMinorOffsetConfig](#) ([edma\\_tcd\\_t](#) \*tcd, const [edma\\_minor\\_offset\\_config\\_t](#) \*config)  
*Configures the eDMA TCD minor offset feature.*
- void [EDMA\\_TcdSetChannelLink](#) ([edma\\_tcd\\_t](#) \*tcd, [edma\\_channel\\_link\\_type\\_t](#) linkType, uint32\_t linkedChannel)  
*Sets the channel link for the eDMA TCD.*
- static void [EDMA\\_TcdSetBandWidth](#) ([edma\\_tcd\\_t](#) \*tcd, [edma\\_bandwidth\\_t](#) bandWidth)  
*Sets the bandwidth for the eDMA TCD.*
- void [EDMA\\_TcdSetModulo](#) ([edma\\_tcd\\_t](#) \*tcd, [edma\\_modulo\\_t](#) srcModulo, [edma\\_modulo\\_t](#) destModulo)  
*Sets the source modulo and the destination modulo for the eDMA TCD.*
- static void [EDMA\\_TcdEnableAutoStopRequest](#) ([edma\\_tcd\\_t](#) \*tcd, bool enable)  
*Sets the auto stop request for the eDMA TCD.*
- void [EDMA\\_TcdEnableInterrupts](#) ([edma\\_tcd\\_t](#) \*tcd, uint32\_t mask)  
*Enables the interrupt source for the eDMA TCD.*
- void [EDMA\\_TcdDisableInterrupts](#) ([edma\\_tcd\\_t](#) \*tcd, uint32\_t mask)  
*Disables the interrupt source for the eDMA TCD.*
- void [EDMA\\_TcdSetMajorOffsetConfig](#) ([edma\\_tcd\\_t](#) \*tcd, int32\_t sourceOffset, int32\_t destOffset)  
*Configures the eDMA TCD major offset feature.*

## eDMA Channel Transfer Operation

- static void [EDMA\\_EnableChannelRequest](#) ([DMA\\_Type](#) \*base, uint32\_t channel)  
*Enables the eDMA hardware channel request.*
- static void [EDMA\\_DisableChannelRequest](#) ([DMA\\_Type](#) \*base, uint32\_t channel)  
*Disables the eDMA hardware channel request.*
- static void [EDMA\\_TriggerChannelStart](#) ([DMA\\_Type](#) \*base, uint32\_t channel)  
*Starts the eDMA transfer by using the software trigger.*

## eDMA Channel Status Operation

- uint32\_t [EDMA\\_GetRemainingMajorLoopCount](#) ([DMA\\_Type](#) \*base, uint32\_t channel)  
*Gets the remaining major loop count from the eDMA current channel TCD.*
- static uint32\_t [EDMA\\_GetErrorStatusFlags](#) ([DMA\\_Type](#) \*base)  
*Gets the eDMA channel error status flags.*
- uint32\_t [EDMA\\_GetChannelStatusFlags](#) ([DMA\\_Type](#) \*base, uint32\_t channel)  
*Gets the eDMA channel status flags.*
- void [EDMA\\_ClearChannelStatusFlags](#) ([DMA\\_Type](#) \*base, uint32\_t channel, uint32\_t mask)  
*Clears the eDMA channel status flags.*

## eDMA Transactional Operation

- void [EDMA\\_CreateHandle](#) ([edma\\_handle\\_t](#) \*handle, [DMA\\_Type](#) \*base, uint32\_t channel)  
*Creates the eDMA handle.*

- void `EDMA_InstallTCDMemory` (`edma_handle_t` \*handle, `edma_tcd_t` \*tcdPool, `uint32_t` tcdSize)  
*Installs the TCDs memory pool into the eDMA handle.*
- void `EDMA_SetCallback` (`edma_handle_t` \*handle, `edma_callback` callback, void \*userData)  
*Installs a callback function for the eDMA transfer.*
- void `EDMA_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` transferType)  
*Prepares the eDMA transfer structure.*
- `status_t` `EDMA_SubmitTransfer` (`edma_handle_t` \*handle, const `edma_transfer_config_t` \*config)  
*Submits the eDMA transfer request.*
- void `EDMA_StartTransfer` (`edma_handle_t` \*handle)  
*eDMA starts transfer.*
- void `EDMA_StopTransfer` (`edma_handle_t` \*handle)  
*eDMA stops transfer.*
- void `EDMA_AbortTransfer` (`edma_handle_t` \*handle)  
*eDMA aborts transfer.*
- static `uint32_t` `EDMA_GetUnusedTCDDNumber` (`edma_handle_t` \*handle)  
*Get unused TCD slot number.*
- static `uint32_t` `EDMA_GetNextTCDDAddress` (`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.*

## 18.3 Data Structure Documentation

### 18.3.1 struct `edma_config_t`

#### Data Fields

- bool `enableContinuousLinkMode`  
*Enable (true) continuous link mode.*
- bool `enableHaltOnError`  
*Enable (true) transfer halt on error.*
- bool `enableRoundRobinArbitration`  
*Enable (true) round robin channel arbitration method or fixed priority arbitration is used for channel selection.*
- bool `enableDebugMode`  
*Enable(true) eDMA debug mode.*

#### Field Documentation

##### (1) bool `edma_config_t::enableContinuousLinkMode`

Upon minor loop completion, the channel activates again if that channel has a minor loop channel link enabled and the link channel is itself.

**(2) bool edma\_config\_t::enableHaltOnError**

Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

**(3) bool edma\_config\_t::enableDebugMode**

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

**18.3.2 struct edma\_transfer\_config\_t**

This structure configures the source/destination transfer attribute.

**Data Fields**

- uint32\_t [srcAddr](#)  
*Source data address.*
- uint32\_t [destAddr](#)  
*Destination data address.*
- [edma\\_transfer\\_size\\_t srcTransferSize](#)  
*Source data transfer size.*
- [edma\\_transfer\\_size\\_t destTransferSize](#)  
*Destination data transfer size.*
- int16\_t [srcOffset](#)  
*Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.*
- int16\_t [destOffset](#)  
*Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.*
- uint32\_t [minorLoopBytes](#)  
*Bytes to transfer in a minor loop.*
- uint32\_t [majorLoopCounts](#)  
*Major loop iteration count.*

## Field Documentation

- (1) `uint32_t edma_transfer_config_t::srcAddr`
- (2) `uint32_t edma_transfer_config_t::destAddr`
- (3) `edma_transfer_size_t edma_transfer_config_t::srcTransferSize`
- (4) `edma_transfer_size_t edma_transfer_config_t::destTransferSize`
- (5) `int16_t edma_transfer_config_t::srcOffset`
- (6) `int16_t edma_transfer_config_t::destOffset`
- (7) `uint32_t edma_transfer_config_t::majorLoopCounts`

### 18.3.3 struct edma\_channel\_Preemption\_config\_t

#### Data Fields

- bool `enableChannelPreemption`  
*If true: a channel can be suspended by other channel with higher priority.*
- bool `enablePreemptAbility`  
*If true: a channel can suspend other channel with low priority.*
- `uint8_t channelPriority`  
*Channel priority.*

### 18.3.4 struct edma\_minor\_offset\_config\_t

#### Data Fields

- bool `enableSrcMinorOffset`  
*Enable(true) or Disable(false) source minor loop offset.*
- bool `enableDestMinorOffset`  
*Enable(true) or Disable(false) destination minor loop offset.*
- `uint32_t minorOffset`  
*Offset for a minor loop mapping.*

## Field Documentation

- (1) `bool edma_minor_offset_config_t::enableSrcMinorOffset`
- (2) `bool edma_minor_offset_config_t::enableDestMinorOffset`
- (3) `uint32_t edma_minor_offset_config_t::minorOffset`

### 18.3.5 struct edma\_tcd\_t

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

## Data Fields

- `__IO uint32_t SADDR`  
*SADDR register, used to save source address.*
- `__IO uint16_t SOFF`  
*SOFF register, save offset bytes every transfer.*
- `__IO uint16_t ATTR`  
*ATTR register, source/destination transfer size and modulo.*
- `__IO uint32_t NBYTES`  
*Nbytes register, minor loop length in bytes.*
- `__IO uint32_t SLAST`  
*SLAST register.*
- `__IO uint32_t DADDR`  
*DADDR register, used for destination address.*
- `__IO uint16_t DOFF`  
*DOFF register, used for destination offset.*
- `__IO uint16_t CITER`  
*CITER register, current minor loop numbers, for unfinished minor loop.*
- `__IO uint32_t DLAST_SGA`  
*DLASTSGA register, next tcd address used in scatter-gather mode.*
- `__IO uint16_t CSR`  
*CSR register, for TCD control status.*
- `__IO uint16_t BITER`  
*BITER register, begin minor loop count.*



## Field Documentation

(1) `__IO uint16_t edma_tcd_t::CITER`

(2) `__IO uint16_t edma_tcd_t::BITER`

18.3.6 `struct edma_handle_t`

## Data Fields

- `edma_callback callback`  
*Callback function for major count exhausted.*
- `void * userData`  
*Callback function parameter.*
- `DMA_Type * base`  
*eDMA peripheral base address.*
- `edma_tcd_t * tcdPool`  
*Pointer to memory stored TCDs.*
- `uint8_t channel`  
*eDMA channel number.*
- `volatile int8_t header`  
*The first TCD index.*
- `volatile int8_t tail`  
*The last TCD index.*
- `volatile int8_t tcdUsed`  
*The number of used TCD slots.*
- `volatile int8_t tcdSize`  
*The total number of TCD slots in the queue.*
- `uint8_t flags`  
*The status of the current channel.*

## Field Documentation

(1) `edma_callback edma_handle_t::callback`

(2) `void* edma_handle_t::userData`

(3) `DMA_Type* edma_handle_t::base`

(4) `edma_tcd_t* edma_handle_t::tcdPool`

(5) `uint8_t edma_handle_t::channel`

(6) `volatile int8_t edma_handle_t::header`

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

(7) `volatile int8_t edma_handle_t::tail`

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

(8) `volatile int8_t edma_handle_t::tcdUsed`

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

(9) `volatile int8_t edma_handle_t::tcdSize`

(10) `uint8_t edma_handle_t::flags`

## 18.4 Macro Definition Documentation

### 18.4.1 `#define FSL_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 3))`

Version 2.4.3.

## 18.5 Typedef Documentation

### 18.5.1 `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. |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## 18.6 Enumeration Type Documentation

### 18.6.1 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.

### 18.6.2 enum edma\_modulo\_t

Enumerator

- kEDMA\_ModuloDisable* Disable modulo.
- kEDMA\_Modulo2bytes* Circular buffer size is 2 bytes.
- kEDMA\_Modulo4bytes* Circular buffer size is 4 bytes.
- kEDMA\_Modulo8bytes* Circular buffer size is 8 bytes.
- kEDMA\_Modulo16bytes* Circular buffer size is 16 bytes.
- kEDMA\_Modulo32bytes* Circular buffer size is 32 bytes.
- kEDMA\_Modulo64bytes* Circular buffer size is 64 bytes.
- kEDMA\_Modulo128bytes* Circular buffer size is 128 bytes.
- kEDMA\_Modulo256bytes* Circular buffer size is 256 bytes.
- kEDMA\_Modulo512bytes* Circular buffer size is 512 bytes.
- kEDMA\_Modulo1Kbytes* Circular buffer size is 1 K bytes.
- kEDMA\_Modulo2Kbytes* Circular buffer size is 2 K bytes.
- kEDMA\_Modulo4Kbytes* Circular buffer size is 4 K bytes.
- kEDMA\_Modulo8Kbytes* Circular buffer size is 8 K bytes.
- kEDMA\_Modulo16Kbytes* Circular buffer size is 16 K bytes.
- kEDMA\_Modulo32Kbytes* Circular buffer size is 32 K bytes.
- kEDMA\_Modulo64Kbytes* Circular buffer size is 64 K bytes.
- kEDMA\_Modulo128Kbytes* Circular buffer size is 128 K bytes.
- kEDMA\_Modulo256Kbytes* Circular buffer size is 256 K bytes.
- kEDMA\_Modulo512Kbytes* Circular buffer size is 512 K bytes.
- kEDMA\_Modulo1Mbytes* Circular buffer size is 1 M bytes.
- kEDMA\_Modulo2Mbytes* Circular buffer size is 2 M bytes.
- kEDMA\_Modulo4Mbytes* Circular buffer size is 4 M bytes.

*kEDMA\_Modulo8Mbytes* Circular buffer size is 8 M bytes.  
*kEDMA\_Modulo16Mbytes* Circular buffer size is 16 M bytes.  
*kEDMA\_Modulo32Mbytes* Circular buffer size is 32 M bytes.  
*kEDMA\_Modulo64Mbytes* Circular buffer size is 64 M bytes.  
*kEDMA\_Modulo128Mbytes* Circular buffer size is 128 M bytes.  
*kEDMA\_Modulo256Mbytes* Circular buffer size is 256 M bytes.  
*kEDMA\_Modulo512Mbytes* Circular buffer size is 512 M bytes.  
*kEDMA\_Modulo1Gbytes* Circular buffer size is 1 G bytes.  
*kEDMA\_Modulo2Gbytes* Circular buffer size is 2 G bytes.

### 18.6.3 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.

### 18.6.4 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.

### 18.6.5 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

### 18.6.6 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.

### 18.6.7 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.

### 18.6.8 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.

### 18.6.9 anonymous enum

Enumerator

*kStatus\_EDMA\_QueueFull* TCD queue is full.  
*kStatus\_EDMA\_Busy* Channel is busy and can't handle the transfer request.

## 18.7 Function Documentation

### 18.7.1 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.

### 18.7.2 void EDMA\_Deinit ( DMA\_Type \* *base* )

This function gates the eDMA clock.

## Parameters

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

### 18.7.3 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.       |

### 18.7.4 void EDMA\_GetDefaultConfig ( edma\_config\_t \* *config* )

This function sets the configuration structure to default values. The default configuration is set to the following values.

```
* config.enableContinuousLinkMode = false;
* config.enableHaltOnError = true;
* config.enableRoundRobinArbitration = false;
* config.enableDebugMode = false;
*
```

## Parameters

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

### 18.7.5 static void EDMA\_EnableContinuousChannelLinkMode ( DMA\_Type \* *base*, bool *enable* ) [inline], [static]

## Note

Do not use continuous link mode with a channel linking to itself if there is only one minor loop iteration per service request, for example, if the channel's NBYTES value is the same as either the source or destination size. The same data transfer profile can be achieved by simply increasing the NBYTES value, which provides more efficient, faster processing.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | EDMA peripheral base address.     |
| <i>enable</i> | true is enable, false is disable. |

### 18.7.6 static void EDMA\_EnableMinorLoopMapping ( DMA\_Type \* *base*, bool *enable* ) [inline], [static]

The TCDn.word2 is redefined to include individual enable fields, an offset field, and the NBYTES field.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | EDMA peripheral base address.     |
| <i>enable</i> | true is enable, false is disable. |

### 18.7.7 void EDMA\_ResetChannel ( DMA\_Type \* *base*, uint32\_t *channel* )

This function sets TCD registers for this channel to default values.

## Parameters

---

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

## Note

This function must not be called while the channel transfer is ongoing or it causes unpredictable results.

This function enables the auto stop request feature.

### 18.7.8 void EDMA\_SetTransferConfig ( DMA\_Type \* *base*, uint32\_t *channel*, const edma\_transfer\_config\_t \* *config*, edma\_tcd\_t \* *nextTcd* )

This function configures the transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the TCD address.

Example:

```
* edma_transfer_t config;
* edma_tcd_t tcd;
* config.srcAddr = ..;
* config.destAddr = ..;
* ...
* EDMA_SetTransferConfig(DMA0, channel, &config, &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.

### 18.7.9 void EDMA\_SetMinorOffsetConfig ( DMA\_Type \* *base*, uint32\_t *channel*, const edma\_minor\_offset\_config\_t \* *config* )

The minor offset means that the signed-extended value is added to the source address or destination address after each minor loop.



## Parameters

|                |                                                        |
|----------------|--------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                          |
| <i>channel</i> | eDMA channel number.                                   |
| <i>config</i>  | A pointer to the minor offset configuration structure. |

### 18.7.10 void EDMA\_SetChannelPreemptionConfig ( DMA\_Type \* *base*, uint32\_t *channel*, const edma\_channel\_Preemption\_config\_t \* *config* )

This function configures the channel preemption attribute and the priority of the channel.

## Parameters

|                |                                                              |
|----------------|--------------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                                |
| <i>channel</i> | eDMA channel number                                          |
| <i>config</i>  | A pointer to the channel preemption configuration structure. |

### 18.7.11 void EDMA\_SetChannelLink ( DMA\_Type \* *base*, uint32\_t *channel*, edma\_channel\_link\_type\_t *linkType*, uint32\_t *linkedChannel* )

This function configures either the minor link or the major link mode. The minor link means that the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

## Parameters

|                 |                                                                                                                                                                                  |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>     | eDMA peripheral base address.                                                                                                                                                    |
| <i>channel</i>  | eDMA channel number.                                                                                                                                                             |
| <i>linkType</i> | A channel link type, which can be one of the following: <ul style="list-style-type: none"> <li>• kEDMA_LinkNone</li> <li>• kEDMA_MinorLink</li> <li>• kEDMA_MajorLink</li> </ul> |

|                      |                            |
|----------------------|----------------------------|
| <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.

### 18.7.12 void EDMA\_SetBandWidth ( DMA\_Type \* *base*, uint32\_t *channel*, edma\_bandwidth\_t *bandWidth* )

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

## Parameters

|                  |                                                                                                                                                                                                               |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | eDMA peripheral base address.                                                                                                                                                                                 |
| <i>channel</i>   | eDMA channel number.                                                                                                                                                                                          |
| <i>bandWidth</i> | A bandwidth setting, which can be one of the following: <ul style="list-style-type: none"> <li>• kEDMABandwidthStallNone</li> <li>• kEDMABandwidthStall4Cycle</li> <li>• kEDMABandwidthStall8Cycle</li> </ul> |

### 18.7.13 void EDMA\_SetModulo ( DMA\_Type \* *base*, uint32\_t *channel*, edma\_modulo\_t *srcModulo*, edma\_modulo\_t *destModulo* )

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

## Parameters

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

|                   |                             |
|-------------------|-----------------------------|
| <i>srcModulo</i>  | A source modulo value.      |
| <i>destModulo</i> | A destination modulo value. |

**18.7.14 static void EDMA\_EnableAsyncRequest ( DMA\_Type \* *base*, uint32\_t *channel*, bool *enable* ) [inline], [static]**

Parameters

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                    |
| <i>channel</i> | eDMA channel number.                             |
| <i>enable</i>  | The command to enable (true) or disable (false). |

**18.7.15 static void EDMA\_EnableAutoStopRequest ( DMA\_Type \* *base*, uint32\_t *channel*, bool *enable* ) [inline], [static]**

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                    |
| <i>channel</i> | eDMA channel number.                             |
| <i>enable</i>  | The command to enable (true) or disable (false). |

**18.7.16 void EDMA\_EnableChannelInterrupts ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *mask* )**

Parameters

|                |                                                                                                                  |
|----------------|------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                                                                                    |
| <i>channel</i> | eDMA channel number.                                                                                             |
| <i>mask</i>    | The mask of interrupt source to be set. Users need to use the defined <code>edma_interrupt_enable_t</code> type. |

**18.7.17 void EDMA\_DisableChannelInterrupts ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *mask* )**

Parameters

|                |                                                                                                        |
|----------------|--------------------------------------------------------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                                                                          |
| <i>channel</i> | eDMA channel number.                                                                                   |
| <i>mask</i>    | The mask of the interrupt source to be set. Use the defined <code>edma_interrupt_enable_t</code> type. |

### 18.7.18 void EDMA\_SetMajorOffsetConfig ( DMA\_Type \* *base*, uint32\_t *channel*, int32\_t *sourceOffset*, int32\_t *destOffset* )

Adjustment value added to the source address at the completion of the major iteration count

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>base</i>         | eDMA peripheral base address.                                                       |
| <i>channel</i>      | edma channel number.                                                                |
| <i>sourceOffset</i> | source address offset will be applied to source address after major loop done.      |
| <i>destOffset</i>   | destination address offset will be applied to source address after major loop done. |

### 18.7.19 void EDMA\_TcdReset ( edma\_tcd\_t \* *tcd* )

This function sets all fields for this TCD structure to default value.

Parameters

|            |                               |
|------------|-------------------------------|
| <i>tcd</i> | Pointer to the TCD structure. |
|------------|-------------------------------|

Note

This function enables the auto stop request feature.

### 18.7.20 void EDMA\_TcdSetTransferConfig ( edma\_tcd\_t \* *tcd*, const edma\_transfer\_config\_t \* *config*, edma\_tcd\_t \* *nextTcd* )

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The TCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```

* edma_transfer_t config = {
* ...
* }
* edma_tcd_t tcd __aligned(32);
* edma_tcd_t nextTcd __aligned(32);
* EDMA_TcdSetTransferConfig(&tcd, &config, &nextTcd);
*

```

## Parameters

|                |                                                                                                          |
|----------------|----------------------------------------------------------------------------------------------------------|
| <i>tcd</i>     | Pointer to the TCD structure.                                                                            |
| <i>config</i>  | Pointer to eDMA transfer configuration structure.                                                        |
| <i>nextTcd</i> | Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature. |

## Note

TCD address should be 32 bytes aligned or it causes an eDMA error.

If the nextTcd is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA\_TcdReset.

### 18.7.21 void EDMA\_TcdSetMinorOffsetConfig ( edma\_tcd\_t \* *tcd*, const edma\_minor\_offset\_config\_t \* *config* )

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

## Parameters

|               |                                                        |
|---------------|--------------------------------------------------------|
| <i>tcd</i>    | A point to the TCD structure.                          |
| <i>config</i> | A pointer to the minor offset configuration structure. |

### 18.7.22 void EDMA\_TcdSetChannelLink ( edma\_tcd\_t \* *tcd*, edma\_channel\_link\_type\_t *linkType*, uint32\_t *linkedChannel* )

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

## Note

Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

## Parameters

|                      |                                                                                                                                                               |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>tcd</i>           | Point to the TCD structure.                                                                                                                                   |
| <i>linkType</i>      | Channel link type, it can be one of: <ul style="list-style-type: none"> <li>• kEDMA_LinkNone</li> <li>• kEDMA_MinorLink</li> <li>• kEDMA_MajorLink</li> </ul> |
| <i>linkedChannel</i> | The linked channel number.                                                                                                                                    |

### 18.7.23 static void EDMA\_TcdSetBandWidth ( edma\_tcd\_t \* *tcd*, edma\_bandwidth\_t *bandWidth* ) [inline], [static]

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

## Parameters

|                  |                                                                                                                                                                                                               |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>tcd</i>       | A pointer to the TCD structure.                                                                                                                                                                               |
| <i>bandWidth</i> | A bandwidth setting, which can be one of the following: <ul style="list-style-type: none"> <li>• kEDMABandwidthStallNone</li> <li>• kEDMABandwidthStall4Cycle</li> <li>• kEDMABandwidthStall8Cycle</li> </ul> |

### 18.7.24 void EDMA\_TcdSetModulo ( edma\_tcd\_t \* *tcd*, edma\_modulo\_t *srcModulo*, edma\_modulo\_t *destModulo* )

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

## Parameters

|            |                                 |
|------------|---------------------------------|
| <i>tcd</i> | A pointer to the TCD structure. |
|------------|---------------------------------|

|                   |                             |
|-------------------|-----------------------------|
| <i>srcModulo</i>  | A source modulo value.      |
| <i>destModulo</i> | A destination modulo value. |

### 18.7.25 static void EDMA\_TcdEnableAutoStopRequest ( edma\_tcd\_t \* *tcd*, bool *enable* ) [inline], [static]

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

|               |                                                  |
|---------------|--------------------------------------------------|
| <i>tcd</i>    | A pointer to the TCD structure.                  |
| <i>enable</i> | The command to enable (true) or disable (false). |

### 18.7.26 void EDMA\_TcdEnableInterrupts ( edma\_tcd\_t \* *tcd*, uint32\_t *mask* )

Parameters

|             |                                                                                                     |
|-------------|-----------------------------------------------------------------------------------------------------|
| <i>tcd</i>  | Point to the TCD structure.                                                                         |
| <i>mask</i> | The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type. |

### 18.7.27 void EDMA\_TcdDisableInterrupts ( edma\_tcd\_t \* *tcd*, uint32\_t *mask* )

Parameters

|             |                                                                                                     |
|-------------|-----------------------------------------------------------------------------------------------------|
| <i>tcd</i>  | Point to the TCD structure.                                                                         |
| <i>mask</i> | The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type. |

### 18.7.28 void EDMA\_TcdSetMajorOffsetConfig ( edma\_tcd\_t \* *tcd*, int32\_t *sourceOffset*, int32\_t *destOffset* )

Adjustment value added to the source address at the completion of the major iteration count

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>tcd</i>          | A point to the TCD structure.                                                       |
| <i>sourceOffset</i> | source address offset will be applied to source address after major loop done.      |
| <i>destOffset</i>   | destination address offset will be applied to source address after major loop done. |

### 18.7.29 static void EDMA\_EnableChannelRequest ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This function enables the hardware channel request.

Parameters

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

### 18.7.30 static void EDMA\_DisableChannelRequest ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This function disables the hardware channel request.

Parameters

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

### 18.7.31 static void EDMA\_TriggerChannelStart ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This function starts a minor loop transfer.

Parameters

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



### 18.7.32 `uint32_t` `EDMA_GetRemainingMajorLoopCount` ( `DMA_Type` \* *base*, `uint32_t` *channel* )

This function checks the TCD (Task Control Descriptor) status for a specified eDMA channel and returns the number of major loop count that has not finished.

Parameters

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

Returns

Major loop count which has not been transferred yet for the current TCD.

Note

1. This function can only be used to get unfinished major loop count of transfer without the next TCD, or it might be inaccurate.
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)

**18.7.33 static uint32\_t EDMA\_GetErrorStatusFlags ( DMA\_Type \* *base* )  
[inline], [static]**

Parameters

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

Returns

The mask of error status flags. Users need to use the \_edma\_error\_status\_flags type to decode the return variables.

**18.7.34 uint32\_t EDMA\_GetChannelStatusFlags ( DMA\_Type \* *base*, uint32\_t *channel* )**

## Parameters

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

## Returns

The mask of channel status flags. Users need to use the `_edma_channel_status_flags` type to decode the return variables.

### 18.7.35 void EDMA\_ClearChannelStatusFlags ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *mask* )

## Parameters

|                |                                                                                                                       |
|----------------|-----------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                                                                                         |
| <i>channel</i> | eDMA channel number.                                                                                                  |
| <i>mask</i>    | The mask of channel status to be cleared. Users need to use the defined <code>_edma_channel_status_flags</code> type. |

### 18.7.36 void EDMA\_CreateHandle ( edma\_handle\_t \* *handle*, DMA\_Type \* *base*, uint32\_t *channel* )

This function is called if using the transactional API for eDMA. This function initializes the internal state of the eDMA handle.

## Parameters

|                |                                                                               |
|----------------|-------------------------------------------------------------------------------|
| <i>handle</i>  | eDMA handle pointer. The eDMA handle stores callback function and parameters. |
| <i>base</i>    | eDMA peripheral base address.                                                 |
| <i>channel</i> | eDMA channel number.                                                          |

### 18.7.37 void EDMA\_InstallTCMemory ( edma\_handle\_t \* *handle*, edma\_tcd\_t \* *tcdPool*, uint32\_t *tcdSize* )

This function is called after the `EDMA_CreateHandle` to use scatter/gather feature. This function shall only be used while users need to use scatter gather mode. Scatter gather mode enables EDMA to load a new transfer control block (tcd) in hardware, and automatically reconfigure that DMA channel for a

new transfer. Users need to prepare tcd memory and also configure tcds using interface `EDMA_SubmitTransfer`.

## Parameters

|                |                                                           |
|----------------|-----------------------------------------------------------|
| <i>handle</i>  | eDMA handle pointer.                                      |
| <i>tcdPool</i> | A memory pool to store TCDs. It must be 32 bytes aligned. |
| <i>tcdSize</i> | The number of TCD slots.                                  |

### 18.7.38 void EDMA\_SetCallback ( edma\_handle\_t \* *handle*, edma\_callback *callback*, void \* *userData* )

This callback is called in the eDMA IRQ handler. Use the callback to do something after the current major loop transfer completes. This function will be called every time one tcd finished transfer.

## Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>handle</i>   | eDMA handle pointer.                   |
| <i>callback</i> | eDMA callback function pointer.        |
| <i>userData</i> | A parameter for the callback function. |

### 18.7.39 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.

## Parameters

|                         |                                                           |
|-------------------------|-----------------------------------------------------------|
| <i>config</i>           | The user configuration structure of type edma_transfer_t. |
| <i>srcAddr</i>          | eDMA transfer source address.                             |
| <i>srcWidth</i>         | eDMA transfer source address width(bytes).                |
| <i>srcOffset</i>        | source address offset.                                    |
| <i>destAddr</i>         | eDMA transfer destination address.                        |
| <i>destWidth</i>        | eDMA transfer destination address width(bytes).           |
| <i>destOffset</i>       | destination address offset.                               |
| <i>bytesEachRequest</i> | eDMA transfer bytes per channel request.                  |
| <i>transferBytes</i>    | eDMA transfer bytes to be transferred.                    |

## Note

The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

**18.7.40 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 transferType )**

This function prepares the transfer configuration structure according to the user input.

## Parameters

|                         |                                                           |
|-------------------------|-----------------------------------------------------------|
| <i>config</i>           | The user configuration structure of type edma_transfer_t. |
| <i>srcAddr</i>          | eDMA transfer source address.                             |
| <i>srcWidth</i>         | eDMA transfer source address width(bytes).                |
| <i>destAddr</i>         | eDMA transfer destination address.                        |
| <i>destWidth</i>        | eDMA transfer destination address width(bytes).           |
| <i>bytesEachRequest</i> | eDMA transfer bytes per channel request.                  |
| <i>transferBytes</i>    | eDMA transfer bytes to be transferred.                    |
| <i>transferType</i>     | eDMA transfer type.                                       |

## Note

The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

**18.7.41 status\_t EDMA\_SubmitTransfer ( edma\_handle\_t \* handle, const edma\_transfer\_config\_t \* config )**

This function submits the eDMA transfer request according to the transfer configuration structure. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA\_InstallTCDMemory before.

## Parameters

|               |                                                   |
|---------------|---------------------------------------------------|
| <i>handle</i> | eDMA handle pointer.                              |
| <i>config</i> | Pointer to eDMA transfer configuration structure. |

## Return values

|                                |                                                                     |
|--------------------------------|---------------------------------------------------------------------|
| <i>kStatus_EDMA_Success</i>    | It means submit transfer request succeed.                           |
| <i>kStatus_EDMA_Queue-Full</i> | It means TCD queue is full. Submit transfer request is not allowed. |
| <i>kStatus_EDMA_Busy</i>       | It means the given channel is busy, need to submit request later.   |

**18.7.42 void EDMA\_StartTransfer ( edma\_handle\_t \* *handle* )**

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

## Parameters

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

**18.7.43 void EDMA\_StopTransfer ( edma\_handle\_t \* *handle* )**

This function disables the channel request to pause the transfer. Users can call [EDMA\\_StartTransfer\(\)](#) again to resume the transfer.

## Parameters

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

**18.7.44 void EDMA\_AbortTransfer ( edma\_handle\_t \* *handle* )**

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

Parameters

|               |                     |
|---------------|---------------------|
| <i>handle</i> | DMA handle pointer. |
|---------------|---------------------|

#### 18.7.45 **static uint32\_t EDMA\_GetUnusedTCDNumber ( edma\_handle\_t \* *handle* )** **[inline], [static]**

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.

Parameters

|               |                     |
|---------------|---------------------|
| <i>handle</i> | DMA handle pointer. |
|---------------|---------------------|

Returns

The unused tcd slot number.

#### 18.7.46 **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.

#### 18.7.47 **void EDMA\_HandleIRQ ( edma\_handle\_t \* *handle* )**

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Note: For the case using TCD queue, when the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor (if scatter/gather is enabled).



For instance, when the time interrupt of TCD[0] happens, the TCD[1] has already been loaded into the eDMA engine. As *sga* and *sga\_index* are calculated based on the *DLAST\_SGA* bitfield lies in the *TCD\_CSR* register, the *sga\_index* in this case should be 2 (*DLAST\_SGA* of TCD[1] stores the address of TCD[2]). Thus, the "tcdUsed" updated should be (*tcdUsed* - 2U) which indicates the number of TCDs can be loaded in the memory pool (because TCD[0] and TCD[1] have been loaded into the eDMA engine at this point already.).

For the last two continuous ISRs in a scatter/gather process, they both load the last TCD (The last ISR does not load a new TCD) from the memory pool to the eDMA engine when major loop completes. Therefore, ensure that the header and *tcdUsed* updated are identical for them. *tcdUsed* are both 0 in this case as no TCD to be loaded.

See the "eDMA basic data flow" in the eDMA Functional description section of the Reference Manual for further details.

### Parameters

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

# Chapter 19

## ENC: Quadrature Encoder/Decoder

### 19.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Quadrature Encoder/Decoder (ENC) module of MCUXpresso SDK devices.

This section describes the programming interface of the ENC Peripheral driver. The ENC driver configures the ENC module and provides a functional interface for the user to build the ENC application.

### 19.2 Function groups

#### 19.2.1 Initialization and De-initialization

This function group initializes default configuration structure for the ENC counter and initializes ENC counter with the normal configuration and de-initialize ENC module. Some APIs are also created to control the features.

#### 19.2.2 Status

This function group get/clear the ENC status.

#### 19.2.3 Interrupts

This function group enable/disable the ENC interrupts.

#### 19.2.4 Value Operation

This function group get the counter/hold value of positions.

### 19.3 Typical use case

#### 19.3.1 Polling Configuration

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

### Data Structures

- struct [enc\\_config\\_t](#)

- Define user configuration structure for ENC module. [More...](#)
- struct `enc_self_test_config_t`  
Define configuration structure for self test module. [More...](#)

## Enumerations

- enum `_enc_interrupt_enable` {  
`kENC_HOMETransitionInterruptEnable = (1U << 0U),`  
`kENC_INDEXPulseInterruptEnable = (1U << 1U),`  
`kENC_WatchdogTimeoutInterruptEnable = (1U << 2U),`  
`kENC_PositionCompareInterruptEnable = (1U << 3U),`  
`kENC_SimultBothPhaseChangeInterruptEnable,`  
`kENC_PositionRollOverInterruptEnable = (1U << 5U),`  
`kENC_PositionRollUnderInterruptEnable = (1U << 6U) }`  
*Interrupt enable/disable mask.*
- enum `_enc_status_flags` {  
`kENC_HOMETransitionFlag = (1U << 0U),`  
`kENC_INDEXPulseFlag = (1U << 1U),`  
`kENC_WatchdogTimeoutFlag = (1U << 2U),`  
`kENC_PositionCompareFlag = (1U << 3U),`  
`kENC_SimultBothPhaseChangeFlag = (1U << 4U),`  
`kENC_PositionRollOverFlag = (1U << 5U),`  
`kENC_PositionRollUnderFlag = (1U << 6U),`  
`kENC_LastCountDirectionFlag = (1U << 7U) }`  
*Status flag mask.*
- enum `_enc_signal_status_flags` {  
`kENC_RawHOMESTatusFlag = ENC_IMR_HOME_MASK,`  
`kENC_RawINDEXStatusFlag = ENC_IMR_INDEX_MASK,`  
`kENC_RawPHBStatusFlag = ENC_IMR_PHB_MASK,`  
`kENC_RawPHAEXStatusFlag = ENC_IMR_PHA_MASK,`  
`kENC_FilteredHOMESTatusFlag = ENC_IMR_FHOM_MASK,`  
`kENC_FilteredINDEXStatusFlag = ENC_IMR_FIND_MASK,`  
`kENC_FilteredPHBStatusFlag = ENC_IMR_FPHB_MASK,`  
`kENC_FilteredPHAStatusFlag = ENC_IMR_FPHA_MASK }`  
*Signal status flag mask.*
- enum `enc_home_trigger_mode_t` {  
`kENC_HOMETriggerDisabled = 0U,`  
`kENC_HOMETriggerOnRisingEdge,`  
`kENC_HOMETriggerOnFallingEdge }`  
*Define HOME signal's trigger mode.*
- enum `enc_index_trigger_mode_t` {  
`kENC_INDEXTriggerDisabled = 0U,`  
`kENC_INDEXTriggerOnRisingEdge,`  
`kENC_INDEXTriggerOnFallingEdge }`  
*Define INDEX signal's trigger mode.*
- enum `enc_decoder_work_mode_t` {  
`kENC_DecoderWorkAsNormalMode = 0U,`

```
kENC_DecoderWorkAsSignalPhaseCountMode }
```

*Define type for decoder work mode.*

- enum `enc_position_match_mode_t` {  
`kENC_POSMATCHOnPositionCounterEqualToComapreValue = 0U`,  
`kENC_POSMATCHOnReadingAnyPositionCounter` }

*Define type for the condition of POSMATCH pulses.*

- enum `enc_revolution_count_condition_t` {  
`kENC_RevolutionCountOnINDEXPulse = 0U`,  
`kENC_RevolutionCountOnRollOverModulus` }

*Define type for determining how the revolution counter (REV) is incremented/decremented.*

- enum `enc_self_test_direction_t` {  
`kENC_SelfTestDirectionPositive = 0U`,  
`kENC_SelfTestDirectionNegative` }

*Define type for direction of self test generated signal.*

## Initialization and De-initialization

- void `ENC_Init` (ENC\_Type \*base, const `enc_config_t` \*config)  
*Initialization for the ENC module.*
- void `ENC_Deinit` (ENC\_Type \*base)  
*De-initialization for the ENC module.*
- void `ENC_GetDefaultConfig` (`enc_config_t` \*config)  
*Get an available pre-defined settings for ENC's configuration.*
- void `ENC_DoSoftwareLoadInitialPositionValue` (ENC\_Type \*base)  
*Load the initial position value to position counter.*
- void `ENC_SetSelfTestConfig` (ENC\_Type \*base, const `enc_self_test_config_t` \*config)  
*Enable and configure the self test function.*
- void `ENC_EnableWatchdog` (ENC\_Type \*base, bool enable)  
*Enable watchdog for ENC module.*
- void `ENC_SetInitialPositionValue` (ENC\_Type \*base, uint32\_t value)  
*Set initial position value for ENC module.*

## Status

- uint32\_t `ENC_GetStatusFlags` (ENC\_Type \*base)  
*Get the status flags.*
- void `ENC_ClearStatusFlags` (ENC\_Type \*base, uint32\_t mask)  
*Clear the status flags.*
- static uint16\_t `ENC_GetSignalStatusFlags` (ENC\_Type \*base)  
*Get the signals' real-time status.*

## Interrupts

- void `ENC_EnableInterrupts` (ENC\_Type \*base, uint32\_t mask)  
*Enable the interrupts.*
- void `ENC_DisableInterrupts` (ENC\_Type \*base, uint32\_t mask)  
*Disable the interrupts.*
- uint32\_t `ENC_GetEnabledInterrupts` (ENC\_Type \*base)  
*Get the enabled interrupts' flags.*

## Value Operation

- uint32\_t [ENC\\_GetPositionValue](#) (ENC\_Type \*base)  
*Get the current position counter's value.*
- uint32\_t [ENC\\_GetHoldPositionValue](#) (ENC\_Type \*base)  
*Get the hold position counter's value.*
- static uint16\_t [ENC\\_GetPositionDifferenceValue](#) (ENC\_Type \*base)  
*Get the position difference counter's value.*
- static uint16\_t [ENC\\_GetHoldPositionDifferenceValue](#) (ENC\_Type \*base)  
*Get the hold position difference counter's value.*
- static uint16\_t [ENC\\_GetRevolutionValue](#) (ENC\_Type \*base)  
*Get the position revolution counter's value.*
- static uint16\_t [ENC\\_GetHoldRevolutionValue](#) (ENC\_Type \*base)  
*Get the hold position revolution counter's value.*

## 19.4 Data Structure Documentation

### 19.4.1 struct enc\_config\_t

#### Data Fields

- bool [enableReverseDirection](#)  
*Enable reverse direction counting.*
- enc\_decoder\_work\_mode\_t [decoderWorkMode](#)  
*Enable signal phase count mode.*
- enc\_home\_trigger\_mode\_t [HOMETriggerMode](#)  
*Enable HOME to initialize position counters.*
- enc\_index\_trigger\_mode\_t [INDEXTriggerMode](#)  
*Enable INDEX to initialize position counters.*
- bool [enableTRIGGERClearPositionCounter](#)  
*Clear POSD, REV, UPOS and LPOS on rising edge of TRIGGER, or not.*
- bool [enableTRIGGERClearHoldPositionCounter](#)  
*Enable update of hold registers on rising edge of TRIGGER, or not.*
- bool [enableWatchdog](#)  
*Enable the watchdog to detect if the target is moving or not.*
- uint16\_t [watchdogTimeoutValue](#)  
*Watchdog timeout count value.*
- uint16\_t [filterCount](#)  
*Input Filter Sample Count.*
- uint16\_t [filterSamplePeriod](#)  
*Input Filter Sample Period.*
- enc\_position\_match\_mode\_t [positionMatchMode](#)  
*The condition of POSMATCH pulses.*
- uint32\_t [positionCompareValue](#)  
*Position compare value.*
- enc\_revolution\_count\_condition\_t [revolutionCountCondition](#)  
*Revolution Counter Modulus Enable.*
- bool [enableModuloCountMode](#)  
*Enable Modulo Counting.*
- uint32\_t [positionModulusValue](#)  
*Position modulus value.*

- uint32\_t [positionInitialValue](#)  
*Position initial value.*

### Field Documentation

- (1) **bool enc\_config\_t::enableReverseDirection**
- (2) **enc\_decoder\_work\_mode\_t enc\_config\_t::decoderWorkMode**
- (3) **enc\_home\_trigger\_mode\_t enc\_config\_t::HOMETriggerMode**
- (4) **enc\_index\_trigger\_mode\_t enc\_config\_t::INDEXTriggerMode**
- (5) **bool enc\_config\_t::enableTRIGGERClearPositionCounter**
- (6) **bool enc\_config\_t::enableWatchdog**
- (7) **uint16\_t enc\_config\_t::watchdogTimeoutValue**

It stores the timeout count for the quadrature decoder module watchdog timer. This field is only available when "enableWatchdog" = true. The available value is a 16-bit unsigned number.

- (8) **uint16\_t enc\_config\_t::filterCount**

This value should be chosen to reduce the probability of noisy samples causing an incorrect transition to be recognized. The value represent the number of consecutive samples that must agree prior to the input filter accepting an input transition. A value of 0x0 represents 3 samples. A value of 0x7 represents 10 samples. The Available range is 0 - 7.

- (9) **uint16\_t enc\_config\_t::filterSamplePeriod**

This value should be set such that the sampling period is larger than the period of the expected noise. This value represents the sampling period (in IPBus clock cycles) of the decoder input signals. The available range is 0 - 255.

- (10) **enc\_position\_match\_mode\_t enc\_config\_t::positionMatchMode**

- (11) **uint32\_t enc\_config\_t::positionCompareValue**

The available value is a 32-bit number.

- (12) **enc\_revolution\_count\_condition\_t enc\_config\_t::revolutionCountCondition**

- (13) **bool enc\_config\_t::enableModuloCountMode**

- (14) **uint32\_t enc\_config\_t::positionModulusValue**

This value would be available only when "enableModuloCountMode" = true. The available value is a 32-bit number.

**(15) uint32\_t enc\_config\_t::positionInitialValue**

The available value is a 32-bit number.

**19.4.2 struct enc\_self\_test\_config\_t**

The self test module provides a quadrature test signal to the inputs of the quadrature decoder module. This is a factory test feature. It is also useful to customers' software development and testing.

**Data Fields**

- [enc\\_self\\_test\\_direction\\_t signalDirection](#)  
*Direction of self test generated signal.*
- [uint16\\_t signalCount](#)  
*Hold the number of quadrature advances to generate.*
- [uint16\\_t signalPeriod](#)  
*Hold the period of quadrature phase in IPBus clock cycles.*

**Field Documentation****(1) enc\_self\_test\_direction\_t enc\_self\_test\_config\_t::signalDirection****(2) uint16\_t enc\_self\_test\_config\_t::signalCount**

The available range is 0 - 255.

**(3) uint16\_t enc\_self\_test\_config\_t::signalPeriod**

The available range is 0 - 31.

**19.5 Enumeration Type Documentation****19.5.1 enum \_enc\_interrupt\_enable**

Enumerator

- kENC\_HOMETransitionInterruptEnable* HOME interrupt enable.
- kENC\_INDEXPulseInterruptEnable* INDEX pulse interrupt enable.
- kENC\_WatchdogTimeoutInterruptEnable* Watchdog timeout interrupt enable.
- kENC\_PositionCompareInterruptEnable* Position compare interrupt enable.
- kENC\_SimultBothPhaseChangeInterruptEnable* Simultaneous PHASEA and PHASEB change interrupt enable.
- kENC\_PositionRollOverInterruptEnable* Roll-over interrupt enable.
- kENC\_PositionRollUnderInterruptEnable* Roll-under interrupt enable.

### 19.5.2 enum \_enc\_status\_flags

These flags indicate the counter's events.

Enumerator

- kENC\_HOMETransitionFlag* HOME signal transition interrupt request.
- kENC\_INDEXPulseFlag* INDEX Pulse Interrupt Request.
- kENC\_WatchdogTimeoutFlag* Watchdog timeout interrupt request.
- kENC\_PositionCompareFlag* Position compare interrupt request.
- kENC\_SimultBothPhaseChangeFlag* Simultaneous PHASEA and PHASEB change interrupt request.
- kENC\_PositionRollOverFlag* Roll-over interrupt request.
- kENC\_PositionRollUnderFlag* Roll-under interrupt request.
- kENC\_LastCountDirectionFlag* Last count was in the up direction, or the down direction.

### 19.5.3 enum \_enc\_signal\_status\_flags

These flags indicate the counter's signal.

Enumerator

- kENC\_RawHOMEStatusFlag* Raw HOME input.
- kENC\_RawINDEXStatusFlag* Raw INDEX input.
- kENC\_RawPHBStatusFlag* Raw PHASEB input.
- kENC\_RawPHAEXStatusFlag* Raw PHASEA input.
- kENC\_FilteredHOMEStatusFlag* The filtered version of HOME input.
- kENC\_FilteredINDEXStatusFlag* The filtered version of INDEX input.
- kENC\_FilteredPHBStatusFlag* The filtered version of PHASEB input.
- kENC\_FilteredPHAStatusFlag* The filtered version of PHASEA input.

### 19.5.4 enum enc\_home\_trigger\_mode\_t

The ENC would count the trigger from HOME signal line.

Enumerator

- kENC\_HOMETriggerDisabled* HOME signal's trigger is disabled.
- kENC\_HOMETriggerOnRisingEdge* Use positive going edge-to-trigger initialization of position counters.
- kENC\_HOMETriggerOnFallingEdge* Use negative going edge-to-trigger initialization of position counters.



### 19.5.5 enum enc\_index\_trigger\_mode\_t

The ENC would count the trigger from INDEX signal line.

Enumerator

*kENC\_INDEXTriggerDisabled* INDEX signal's trigger is disabled.

*kENC\_INDEXTriggerOnRisingEdge* Use positive going edge-to-trigger initialization of position counters.

*kENC\_INDEXTriggerOnFallingEdge* Use negative going edge-to-trigger initialization of position counters.

### 19.5.6 enum enc\_decoder\_work\_mode\_t

The normal work mode uses the standard quadrature decoder with PHASEA and PHASEB. When in signal phase count mode, a positive transition of the PHASEA input generates a count signal while the PHASEB input and the reverse direction control the counter direction. If the reverse direction is not enabled, PHASEB = 0 means counting up and PHASEB = 1 means counting down. Otherwise, the direction is reversed.

Enumerator

*kENC\_DecoderWorkAsNormalMode* Use standard quadrature decoder with PHASEA and PHASEB.

*kENC\_DecoderWorkAsSignalPhaseCountMode* PHASEA input generates a count signal while PHASEB input control the direction.

### 19.5.7 enum enc\_position\_match\_mode\_t

Enumerator

*kENC\_POSMATCHOnPositionCounterEqualToCompareValue* POSMATCH pulses when a match occurs between the position counters (POS) and the compare value (COMP).

*kENC\_POSMATCHOnReadingAnyPositionCounter* POSMATCH pulses when any position counter register is read.

### 19.5.8 enum enc\_revolution\_count\_condition\_t

Enumerator

*kENC\_RevolutionCountOnINDEXPulse* Use INDEX pulse to increment/decrement revolution counter.

*kENC\_RevolutionCountOnRollOverModulus* Use modulus counting roll-over/under to increment/decrement revolution counter.

### 19.5.9 enum enc\_self\_test\_direction\_t

Enumerator

*kENC\_SelfTestDirectionPositive* Self test generates the signal in positive direction.

*kENC\_SelfTestDirectionNegative* Self test generates the signal in negative direction.

## 19.6 Function Documentation

### 19.6.1 void ENC\_Init ( ENC\_Type \* *base*, const enc\_config\_t \* *config* )

This function is to make the initialization for the ENC module. It should be called firstly before any operation to the ENC with the operations like:

- Enable the clock for ENC module.
- Configure the ENC's working attributes.

Parameters

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

### 19.6.2 void ENC\_Deinit ( ENC\_Type \* *base* )

This function is to make the de-initialization for the ENC module. It could be called when ENC is no longer used with the operations like:

- Disable the clock for ENC module.

Parameters

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

### 19.6.3 void ENC\_GetDefaultConfig ( enc\_config\_t \* *config* )

This function initializes the ENC configuration structure with an available settings, the default value are:

```
* config->enableReverseDirection = false;
* config->decoderWorkMode = kENC_DecoderWorkAsNormalMode
;
* config->HOMETriggerMode = kENC_HOMETriggerDisabled;
* config->INDEXTriggerMode = kENC_INDEXTriggerDisabled;
* config->enableTRIGGERClearPositionCounter = false;
* config->enableTRIGGERClearHoldPositionCounter = false;
* config->enableWatchdog = false;
* config->watchdogTimeoutValue = 0U;
* config->filterCount = 0U;
```

```

* config->filterSamplePeriod = 0U;
* config->positionMatchMode =
 kENC_POSMATCHOnPositionCounterEqualToComapreValue;
* config->positionCompareValue = 0xFFFFFFFFFU;
* config->revolutionCountCondition =
 kENC_RevolutionCountOnINDEXPulse;
* config->enableModuloCountMode = false;
* config->positionModulusValue = 0U;
* config->positionInitialValue = 0U;
* config->prescalerValue = kENC_ClockDiv1;
* config->enablePeriodMeasurementFunction = true;
*

```

Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>config</i> | Pointer to a variable of configuration structure. See to "enc_config_t". |
|---------------|--------------------------------------------------------------------------|

**19.6.4 void ENC\_DoSoftwareLoadInitialPositionValue ( ENC\_Type \* *base* )**

This function is to transfer the initial position value (UNIT and LINIT) contents to position counter (UP-OS and LPOS), so that to provide the consistent operation the position counter registers.

Parameters

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

**19.6.5 void ENC\_SetSelfTestConfig ( ENC\_Type \* *base*, const enc\_self\_test\_config\_t \* *config* )**

This function is to enable and configuration the self test function. It controls and sets the frequency of a quadrature signal generator. It provides a quadrature test signal to the inputs of the quadrature decoder module. It is a factory test feature; however, it may be useful to customers' software development and testing.

Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | ENC peripheral base address.                                                                 |
| <i>config</i> | Pointer to configuration structure. See to "enc_self_test_config_t". Pass "NULL" to disable. |

**19.6.6 void ENC\_EnableWatchdog ( ENC\_Type \* *base*, bool *enable* )**

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | ENC peripheral base address      |
| <i>enable</i> | Enables or disables the watchdog |

### 19.6.7 void ENC\_SetInitialPositionValue ( ENC\_Type \* *base*, uint32\_t *value* )

Parameters

|              |                             |
|--------------|-----------------------------|
| <i>base</i>  | ENC peripheral base address |
| <i>value</i> | Positive initial value      |

### 19.6.8 uint32\_t ENC\_GetStatusFlags ( ENC\_Type \* *base* )

Parameters

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

Returns

Mask value of status flags. For available mask, see to "\_enc\_status\_flags".

### 19.6.9 void ENC\_ClearStatusFlags ( ENC\_Type \* *base*, uint32\_t *mask* )

Parameters

|             |                                                                                           |
|-------------|-------------------------------------------------------------------------------------------|
| <i>base</i> | ENC peripheral base address.                                                              |
| <i>mask</i> | Mask value of status flags to be cleared. For available mask, see to "_enc_status_flags". |

### 19.6.10 static uint16\_t ENC\_GetSignalStatusFlags ( ENC\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

Mask value of signals' real-time status. For available mask, see to "\_enc\_signal\_status\_flags"

### 19.6.11 void ENC\_EnableInterrupts ( ENC\_Type \* *base*, uint32\_t *mask* )

## Parameters

|             |                                                                                             |
|-------------|---------------------------------------------------------------------------------------------|
| <i>base</i> | ENC peripheral base address.                                                                |
| <i>mask</i> | Mask value of interrupts to be enabled. For available mask, see to "_enc_interrupt_enable". |

### 19.6.12 void ENC\_DisableInterrupts ( ENC\_Type \* *base*, uint32\_t *mask* )

## Parameters

|             |                                                                                              |
|-------------|----------------------------------------------------------------------------------------------|
| <i>base</i> | ENC peripheral base address.                                                                 |
| <i>mask</i> | Mask value of interrupts to be disabled. For available mask, see to "_enc_interrupt_enable". |

### 19.6.13 uint32\_t ENC\_GetEnabledInterrupts ( ENC\_Type \* *base* )

## Parameters

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

## Returns

Mask value of enabled interrupts.

### 19.6.14 uint32\_t ENC\_GetPositionValue ( ENC\_Type \* *base* )

## Parameters

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

## Returns

Current position counter's value.

### 19.6.15 `uint32_t ENC_GetHoldPositionValue ( ENC_Type * base )`

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

## Parameters

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

## Returns

Hold position counter's value.

### 19.6.16 `static uint16_t ENC_GetPositionDifferenceValue ( ENC_Type * base )` `[inline], [static]`

## Parameters

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

## Returns

The position difference counter's value.

### 19.6.17 `static uint16_t ENC_GetHoldPositionDifferenceValue ( ENC_Type * base )` `[inline], [static]`

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

## Parameters

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

## Returns

Hold position difference counter's value.

**19.6.18 static uint16\_t ENC\_GetRevolutionValue ( ENC\_Type \* *base* ) [inline], [static]**

## Parameters

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

## Returns

The position revolution counter's value.

**19.6.19 static uint16\_t ENC\_GetHoldRevolutionValue ( ENC\_Type \* *base* ) [inline], [static]**

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

## Parameters

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

## Returns

Hold position revolution counter's value.

## Chapter 20

# ENET: Ethernet MAC Driver

### 20.1 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}

### 20.2 Operations of Ethernet MAC Driver

#### 20.2.1 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.

#### 20.2.2 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.

#### 20.2.3 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 the "ENET\_ENHANCEDBUFFERDESCRIPTOR\_MODE" is defined before using this feature.

#### 20.2.4 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



[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\\_GetRxFrame\(\)](#) and [ENET\\_StartTxFrame\(\)](#). 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.

## 20.2.5 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.

## 20.3 Typical use case

### 20.3.1 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\\_rx\\_frame\\_error\\_t](#)  
*Defines the Rx frame error structure. [More...](#)*

- struct `enet_transfer_stats_t`  
Defines the ENET transfer 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_intcoalesce_config_t`  
Defines the interrupt coalescing configure structure. [More...](#)
- struct `enet_avb_config_t`  
Defines the ENET AVB Configure 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_rx_alloc_callback_t)(ENET_Type *base, void *userData, uint8_t ringId)`  
Defines the ENET Rx memory buffer alloc function pointer.
- typedef void `(* enet_rx_free_callback_t)(ENET_Type *base, void *buffer, void *userData, uint8_t ringId)`  
Defines the ENET Rx memory buffer free function pointer.
- typedef void `(* enet_callback_t)(ENET_Type *base, enet_handle_t *handle, uint32_t ringId, enet_event_t event, enet_frame_info_t *frameInfo, void *userData)`  
ENET callback function.
- typedef void `(* enet_isr_ring_t)(ENET_Type *base, enet_handle_t *handle, uint32_t ringId)`  
Define interrupt IRQ handler.

## Enumerations

- enum {  
`kStatus_ENET_InitMemoryFail`,  
`kStatus_ENET_RxFrameError` = MAKE\_STATUS(kStatusGroup\_ENET, 1U),  
`kStatus_ENET_RxFrameFail` = MAKE\_STATUS(kStatusGroup\_ENET, 2U),  
`kStatus_ENET_RxFrameEmpty` = MAKE\_STATUS(kStatusGroup\_ENET, 3U),  
`kStatus_ENET_RxFrameDrop` = MAKE\_STATUS(kStatusGroup\_ENET, 4U),  
`kStatus_ENET_TxFrameOverLen` = MAKE\_STATUS(kStatusGroup\_ENET, 5U),  
`kStatus_ENET_TxFrameBusy` = MAKE\_STATUS(kStatusGroup\_ENET, 6U),  
`kStatus_ENET_TxFrameFail` = MAKE\_STATUS(kStatusGroup\_ENET, 7U) }

- Defines the status return codes for transaction.*

  - enum `enet_mii_mode_t` {  
`kENET_MiiMode = 0U`,  
`kENET_RmiiMode = 1U`,  
`kENET_RgmiiMode = 2U` }

*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`,  
`kENET_MiiSpeed1000M = 2U` }

*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.*
- 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_mii_extend_opcode` {  
`kENET_MiiAddrWrite_C45 = 0U`,  
`kENET_MiiWriteFrame_C45 = 1U`,  
`kENET_MiiReadFrame_C45 = 3U` }

*Define the MII opcode for extended MDIO\_CLAUSES\_45 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`,  
`kENET_ControlSVLANEnable = 0x0400U`,  
`kENET_ControlVLANUseSecondTag = 0x0800U` }

*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_RxFlush2Interrupt = ENET_EIR_RXFLUSH_2_MASK,
kENET_RxFlush1Interrupt = ENET_EIR_RXFLUSH_1_MASK,
kENET_RxFlush0Interrupt = ENET_EIR_RXFLUSH_0_MASK,
kENET_TxFrame2Interrupt = ENET_EIR_TXF2_MASK,
kENET_TxBuffer2Interrupt = ENET_EIR_TXB2_MASK,
kENET_RxFrame2Interrupt = ENET_EIR_RXF2_MASK,
kENET_RxBuffer2Interrupt = ENET_EIR_RXB2_MASK,
kENET_TxFrame1Interrupt = ENET_EIR_TXF1_MASK,
kENET_TxBuffer1Interrupt = ENET_EIR_TXB1_MASK,
kENET_RxFrame1Interrupt = ENET_EIR_RXF1_MASK,
kENET_RxBuffer1Interrupt = ENET_EIR_RXB1_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\_idle\_slope\_t {

```

kENET_IdleSlope1 = 1U,
kENET_IdleSlope2 = 2U,
kENET_IdleSlope4 = 4U,
kENET_IdleSlope8 = 8U,
kENET_IdleSlope16 = 16U,
kENET_IdleSlope32 = 32U,
kENET_IdleSlope64 = 64U,
kENET_IdleSlope128 = 128U,
kENET_IdleSlope256 = 256U,
kENET_IdleSlope384 = 384U,
kENET_IdleSlope512 = 512U,
kENET_IdleSlope640 = 640U,
kENET_IdleSlope768 = 768U,
kENET_IdleSlope896 = 896U,
kENET_IdleSlope1024 = 1024U,
kENET_IdleSlope1152 = 1152U,
kENET_IdleSlope1280 = 1280U,
kENET_IdleSlope1408 = 1408U,
kENET_IdleSlope1536 = 1536U }

```

*Defines certain idle slope for bandwidth fraction.*

- 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.*

## Variables

- const `clock_ip_name_t s_enetClock` []  
*Pointers to enet clocks for each instance.*

## Driver version

- `#define FSL_ENET_DRIVER_VERSION` (`MAKE_VERSION(2, 7, 0)`)  
*Defines the driver version.*

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

Defines the queue number.

- #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  
*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 without VLAN tag.*
- #define `ENET_FRAME_VLAN_TAGLEN` 4U  
*Ethernet single VLAN tag size.*



- #define `ENET_FRAME_CRC_LEN` 4U  
*CRC size in a frame.*
- #define `ENET_FRAME_TX_LEN_LIMITATION(x)` (((x)->RCR & ENET\_RCR\_MAX\_FL\_MASK) >> ENET\_RCR\_MAX\_FL\_SHIFT) - `ENET_FRAME_CRC_LEN`)
- #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`  
*Enet Tx interrupt flag.*
- #define `ENET_RX_INTERRUPT`  
*Enet Rx interrupt flag.*
- #define `ENET_TS_INTERRUPT` ((uint32\_t)kENET\_TsTimerInterrupt | (uint32\_t)kENET\_TsAvailInterrupt)  
*Enet timestamp interrupt flag.*
- #define `ENET_ERR_INTERRUPT`  
*Enet error interrupt flag.*

## Initialization and De-initialization

- void `ENET_GetDefaultConfig` (`enet_config_t` \*config)  
*Gets the ENET default configuration structure.*
- `status_t` `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.*
- `status_t` `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.*
- static void `ENET_StartSMIWrite` (`ENET_Type` \*base, `uint8_t` phyAddr, `uint8_t` regAddr, `enet_mii_write_t` operation, `uint16_t` data)

- Sends the MDIO IEEE802.3 Clause 22 format write command.*

  - static void [ENET\\_StartSMIRead](#) (ENET\_Type \*base, uint8\_t phyAddr, uint8\_t regAddr, [enet\\_mii\\_read\\_t](#) operation)
- Sends the MDIO IEEE802.3 Clause 22 format read command.*

  - [status\\_t ENET\\_MDIOWrite](#) (ENET\_Type \*base, uint8\_t phyAddr, uint8\_t regAddr, uint16\_t data)

*MDIO write with IEEE802.3 Clause 22 format.*
- [status\\_t ENET\\_MDIORead](#) (ENET\_Type \*base, uint8\_t phyAddr, uint8\_t regAddr, uint16\_t \*pData)

*MDIO read with IEEE802.3 Clause 22 format.*
- static void [ENET\\_StartExtC45SMIWriteReg](#) (ENET\_Type \*base, uint8\_t portAddr, uint8\_t devAddr, uint16\_t regAddr)

*Sends the MDIO IEEE802.3 Clause 45 format write register command.*
- static void [ENET\\_StartExtC45SMIWriteData](#) (ENET\_Type \*base, uint8\_t portAddr, uint8\_t devAddr, uint16\_t data)

*Sends the MDIO IEEE802.3 Clause 45 format write data command.*
- static void [ENET\\_StartExtC45SMIReadData](#) (ENET\_Type \*base, uint8\_t portAddr, uint8\_t devAddr)

*Sends the MDIO IEEE802.3 Clause 45 format read data command.*
- [status\\_t ENET\\_MDIOC45Write](#) (ENET\_Type \*base, uint8\_t portAddr, uint8\_t devAddr, uint16\_t regAddr, uint16\_t data)

*MDIO write with IEEE802.3 Clause 45 format.*
- [status\\_t ENET\\_MDIOC45Read](#) (ENET\_Type \*base, uint8\_t portAddr, uint8\_t devAddr, uint16\_t regAddr, uint16\_t \*pData)

*MDIO read with IEEE802.3 Clause 45 format.*
- static void [ENET\\_SetRGMIIIClockDelay](#) (ENET\_Type \*base, bool txEnabled, bool rxEnabled)

*Control the usage of the delayed tx/rx RGMII clock.*

## 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)
- Activates frame reception for multiple rings.*
- 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\\_ring\\_t](#) ISRHandler)  
*Set the second level Rx IRQ handler.*
- void [ENET\\_SetTxISRHandler](#) (ENET\_Type \*base, [enet\\_isr\\_ring\\_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\\_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.*
- void [ENET\\_GetStatistics](#) (ENET\_Type \*base, [enet\\_transfer\\_stats\\_t](#) \*statistics)  
*Gets statistical data in transfer.*
- [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.*
- void [ENET\\_ReclaimTxDescriptor](#) (ENET\_Type \*base, [enet\\_handle\\_t](#) \*handle, uint8\_t ringId)  
*Reclaim tx descriptors.*
- [status\\_t](#) [ENET\\_GetRxFrame](#) (ENET\_Type \*base, [enet\\_handle\\_t](#) \*handle, [enet\\_rx\\_frame\\_struct\\_t](#) \*rxFrame, uint8\_t ringId)  
*Receives one frame in specified BD ring with zero copy.*
- [status\\_t](#) [ENET\\_StartTxFrame](#) (ENET\_Type \*base, [enet\\_handle\\_t](#) \*handle, [enet\\_tx\\_frame\\_struct\\_t](#) \*txFrame, uint8\_t ringId)  
*Sends one frame in specified BD ring with zero copy.*
- void [ENET\\_TransmitIRQHandler](#) (ENET\_Type \*base, [enet\\_handle\\_t](#) \*handle, uint32\_t ringId)  
*The transmit IRQ handler.*
- void [ENET\\_ReceiveIRQHandler](#) (ENET\_Type \*base, [enet\\_handle\\_t](#) \*handle, uint32\_t ringId)  
*The receive IRQ handler.*
- void [ENET\\_CommonFrame1IRQHandler](#) (ENET\_Type \*base)  
*the common IRQ handler for the tx/rx irq handler.*
- void [ENET\\_CommonFrame2IRQHandler](#) (ENET\_Type \*base)  
*the common IRQ handler for the tx/rx 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\\_Ptp1588IRQHandler](#) (ENET\_Type \*base)  
*the common IRQ handler for the 1588 irq handler.*
- void [ENET\\_CommonFrame0IRQHandler](#) (ENET\_Type \*base)  
*the common IRQ handler for the tx/rx/error etc irq handler.*

## 20.4 Data Structure Documentation

### 20.4.1 struct enet\_rx\_bd\_struct\_t

#### Data Fields

- uint16\_t [length](#)  
*Buffer descriptor data length.*
- uint16\_t [control](#)  
*Buffer descriptor control and status.*
- uint32\_t [buffer](#)  
*Data buffer pointer.*

#### Field Documentation

(1) uint16\_t enet\_rx\_bd\_struct\_t::length

(2) uint16\_t enet\_rx\_bd\_struct\_t::control

(3) uint32\_t enet\_rx\_bd\_struct\_t::buffer

### 20.4.2 struct enet\_tx\_bd\_struct\_t

#### Data Fields

- uint16\_t [length](#)  
*Buffer descriptor data length.*
- uint16\_t [control](#)  
*Buffer descriptor control and status.*
- uint32\_t [buffer](#)  
*Data buffer pointer.*

**Field Documentation**

- (1) `uint16_t enet_tx_bd_struct_t::length`
- (2) `uint16_t enet_tx_bd_struct_t::control`
- (3) `uint32_t enet_tx_bd_struct_t::buffer`

**20.4.3 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.*

**Field Documentation**

- (1) `uint32_t enet_data_error_stats_t::statsRxLenGreaterErr`
- (2) `uint32_t enet_data_error_stats_t::statsRxFcsErr`
- (3) `uint32_t enet_data_error_stats_t::statsRxOverRunErr`
- (4) `uint32_t enet_data_error_stats_t::statsRxTruncateErr`

**20.4.4 struct enet\_rx\_frame\_error\_t****Data Fields**

- `bool statsRxTruncateErr: 1`  
*Receive truncate.*
- `bool statsRxOverRunErr: 1`  
*Receive over run.*
- `bool statsRxFcsErr: 1`  
*Receive CRC error.*
- `bool statsRxAlignErr: 1`  
*Receive non-octet alignment.*
- `bool statsRxLenGreaterErr: 1`  
*Receive length greater than RCR[MAX\_FL].*

## Field Documentation

- (1) `bool enet_rx_frame_error_t::statsRxTruncateErr`
- (2) `bool enet_rx_frame_error_t::statsRxOverRunErr`
- (3) `bool enet_rx_frame_error_t::statsRxFcsErr`
- (4) `bool enet_rx_frame_error_t::statsRxAlignErr`
- (5) `bool enet_rx_frame_error_t::statsRxLenGreaterErr`

20.4.5 `struct enet_transfer_stats_t`

## Data Fields

- `uint32_t statsRxFrameCount`  
*Rx frame number.*
- `uint32_t statsRxFrameOk`  
*Good Rx frame number.*
- `uint32_t statsRxCrcErr`  
*Rx frame number with CRC error.*
- `uint32_t statsRxAlignErr`  
*Rx frame number with alignment error.*
- `uint32_t statsRxDropInvalidSFD`  
*Dropped frame number due to invalid SFD.*
- `uint32_t statsRxFifoOverflowErr`  
*Rx FIFO overflow count.*
- `uint32_t statsTxFrameCount`  
*Tx frame number.*
- `uint32_t statsTxFrameOk`  
*Good Tx frame number.*
- `uint32_t statsTxCrcAlignErr`  
*The transmit frame is error.*
- `uint32_t statsTxFifoUnderRunErr`  
*Tx FIFO underrun count.*

## Field Documentation

- (1) `uint32_t enet_transfer_stats_t::statsRxFrameCount`
- (2) `uint32_t enet_transfer_stats_t::statsRxFrameOk`
- (3) `uint32_t enet_transfer_stats_t::statsRxCrcErr`
- (4) `uint32_t enet_transfer_stats_t::statsRxAlignErr`
- (5) `uint32_t enet_transfer_stats_t::statsRxDropInvalidSFD`
- (6) `uint32_t enet_transfer_stats_t::statsRxFifoOverflowErr`
- (7) `uint32_t enet_transfer_stats_t::statsTxFrameCount`
- (8) `uint32_t enet_transfer_stats_t::statsTxFrameOk`
- (9) `uint32_t enet_transfer_stats_t::statsTxCrcAlignErr`
- (10) `uint32_t enet_transfer_stats_t::statsTxFifoUnderRunErr`

### 20.4.6 struct `enet_frame_info_t`

#### Data Fields

- `void * context`  
*User specified data.*

### 20.4.7 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.*

## Field Documentation

- (1) `enet_frame_info_t* enet_tx_dirty_ring_t::txDirtyBase`
- (2) `uint16_t enet_tx_dirty_ring_t::txGenIdx`
- (3) `uint16_t enet_tx_dirty_ring_t::txConsumIdx`
- (4) `uint16_t enet_tx_dirty_ring_t::txRingLen`
- (5) `bool enet_tx_dirty_ring_t::isFull`

20.4.8 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`  
*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.*

### Field Documentation

- (1) `uint16_t enet_buffer_config_t::rxBdNumber`
- (2) `uint16_t enet_buffer_config_t::txBdNumber`
- (3) `uint16_t enet_buffer_config_t::rxBuffSizeAlign`
- (4) `uint16_t enet_buffer_config_t::txBuffSizeAlign`
- (5) `volatile enet_rx_bd_struct_t* enet_buffer_config_t::rxBdStartAddrAlign`
- (6) `volatile enet_tx_bd_struct_t* enet_buffer_config_t::txBdStartAddrAlign`
- (7) `uint8_t* enet_buffer_config_t::rxBufferAlign`
- (8) `uint8_t* enet_buffer_config_t::txBufferAlign`
- (9) `bool enet_buffer_config_t::rxMaintainEnable`
- (10) `bool enet_buffer_config_t::txMaintainEnable`
- (11) `enet_frame_info_t* enet_buffer_config_t::txFrameInfo`

### 20.4.9 struct enet\_intcoalesce\_config\_t

#### Data Fields

- `uint8_t txCoalesceFrameCount` [FSL\_FEATURE\_ENET\_QUEUE]  
*Transmit interrupt coalescing frame count threshold.*
- `uint16_t txCoalesceTimeCount` [FSL\_FEATURE\_ENET\_QUEUE]  
*Transmit interrupt coalescing timer count threshold.*
- `uint8_t rxCoalesceFrameCount` [FSL\_FEATURE\_ENET\_QUEUE]  
*Receive interrupt coalescing frame count threshold.*
- `uint16_t rxCoalesceTimeCount` [FSL\_FEATURE\_ENET\_QUEUE]  
*Receive interrupt coalescing timer count threshold.*

**Field Documentation**

- (1) `uint8_t enet_intcoalesce_config_t::txCoalesceFrameCount[FSL_FEATURE_ENET_QUEUE]`
- (2) `uint16_t enet_intcoalesce_config_t::txCoalesceTimeCount[FSL_FEATURE_ENET_QUEUE]`
- (3) `uint8_t enet_intcoalesce_config_t::rxCoalesceFrameCount[FSL_FEATURE_ENET_QUEUE]`
- (4) `uint16_t enet_intcoalesce_config_t::rxCoalesceTimeCount[FSL_FEATURE_ENET_QUEUE]`

**20.4.10 struct enet\_avb\_config\_t**

This is used for to configure the extended ring 1 and ring 2.

1. The classification match format is  $(CMP3 \ll 12) | (CMP2 \ll 8) | (CMP1 \ll 4) | CMP0$ . composed of four 3-bit compared VLAN priority field `cmp0~cmp3`, `cm0 ~ cmp3` are used in parallel.

If `CMP1,2,3` are not unused, please set them to the same value as `CMP0`.

1. The `idleSlope` is used to calculate the Band Width fraction,  $BW \text{ fraction} = 1 / (1 + 512/idleSlope)$ . For avb configuration, the BW fraction of Class 1 and Class 2 combined must not exceed 0.75.

**Data Fields**

- `uint16_t rxClassifyMatch [FSL_FEATURE_ENET_QUEUE-1]`  
*The classification match value for the ring.*
- `enet_idle_slope_t idleSlope [FSL_FEATURE_ENET_QUEUE-1]`  
*The idle slope for certian bandwidth fraction.*

**Field Documentation**

- (1) `uint16_t enet_avb_config_t::rxClassifyMatch[FSL_FEATURE_ENET_QUEUE-1]`
- (2) `enet_idle_slope_t enet_avb_config_t::idleSlope[FSL_FEATURE_ENET_QUEUE-1]`

**20.4.11 struct enet\_config\_t**

Note:

1. `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.
2. `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.
3. `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\_tx\_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.*
- [enet\\_intcoalesce\\_config\\_t](#) \* [intCoalesceCfg](#)  
*If the interrupt coalesce is not required in the ring n(0,1,2),*

- *please set to NULL.*
- `uint8_t ringNum`  
*Number of used rings.*
- `enet_rx_alloc_callback_t rxBuffAlloc`  
*Callback function to alloc memory, must be provided for zero-copy Rx.*
- `enet_rx_free_callback_t rxBuffFree`  
*Callback function to free memory, must be provided for zero-copy Rx.*
- `enet_callback_t callback`  
*General callback function.*
- `void * userData`  
*Callback function parameter.*

### Field Documentation

#### (1) `uint32_t enet_config_t::macSpecialConfig`

A logical OR of "enet\_special\_control\_flag\_t".

#### (2) `uint32_t enet_config_t::interrupt`

A logical OR of "enet\_interrupt\_enable\_t".

#### (3) `uint16_t enet_config_t::rxMaxFrameLen`

#### (4) `enet_mii_mode_t enet_config_t::miiMode`

#### (5) `enet_mii_speed_t enet_config_t::miiSpeed`

#### (6) `enet_mii_duplex_t enet_config_t::miiDuplex`

#### (7) `uint8_t enet_config_t::rxAccelerConfig`

#### (8) `uint8_t enet_config_t::txAccelerConfig`

#### (9) `uint16_t enet_config_t::pauseDuration`

#### (10) `uint8_t enet_config_t::rxFifoEmptyThreshold`

#### (11) `uint8_t enet_config_t::rxFifoStatEmptyThreshold`

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

#### (12) `uint8_t enet_config_t::rxFifoFullThreshold`

#### (13) `uint8_t enet_config_t::txFifoWatermark`

#### (14) `enet_intcoalesce_config_t* enet_config_t::intCoalesceCfg`

#### (15) `uint8_t enet_config_t::ringNum`

default with 1 – single ring.

- (16) `enet_rx_alloc_callback_t enet_config_t::rxBuffAlloc`
- (17) `enet_rx_free_callback_t enet_config_t::rxBuffFree`
- (18) `enet_callback_t enet_config_t::callback`
- (19) `void* enet_config_t::userData`

#### 20.4.12 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 txConsumIdx`  
*Transmit consume index.*
- volatile `uint16_t txDescUsed`  
*Transmit descriptor used number.*
- `uint16_t txRingLen`  
*Transmit ring length.*

##### Field Documentation

- (1) `volatile enet_tx_bd_struct_t* enet_tx_bd_ring_t::txBdBase`
- (2) `uint16_t enet_tx_bd_ring_t::txGenIdx`
- (3) `uint16_t enet_tx_bd_ring_t::txConsumIdx`
- (4) `volatile uint16_t enet_tx_bd_ring_t::txDescUsed`
- (5) `uint16_t enet_tx_bd_ring_t::txRingLen`

#### 20.4.13 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.*

## Field Documentation

- (1) `volatile enet_rx_bd_struct_t* enet_rx_bd_ring_t::rxBdBase`
- (2) `uint16_t enet_rx_bd_ring_t::rxGenIdx`
- (3) `uint16_t enet_rx_bd_ring_t::rxRingLen`

20.4.14 `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.*
- `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.*
- `enet_rx_alloc_callback_t rxBuffAlloc`  
*Callback function to alloc memory for zero copy Rx.*
- `enet_rx_free_callback_t rxBuffFree`  
*Callback function to free memory for zero copy Rx.*
- `uint8_t multicastCount` [64]  
*Multicast collisions counter.*

## Field Documentation

- (1) `enet_rx_bd_ring_t enet_handle_t::rxBdRing[FSL_FEATURE_ENET_QUEUE]`
- (2) `enet_tx_bd_ring_t enet_handle_t::txBdRing[FSL_FEATURE_ENET_QUEUE]`
- (3) `uint16_t enet_handle_t::rxBuffSizeAlign[FSL_FEATURE_ENET_QUEUE]`
- (4) `uint16_t enet_handle_t::txBuffSizeAlign[FSL_FEATURE_ENET_QUEUE]`
- (5) `bool enet_handle_t::rxMaintainEnable[FSL_FEATURE_ENET_QUEUE]`
- (6) `bool enet_handle_t::txMaintainEnable[FSL_FEATURE_ENET_QUEUE]`
- (7) `uint8_t enet_handle_t::ringNum`
- (8) `enet_callback_t enet_handle_t::callback`
- (9) `void* enet_handle_t::userData`
- (10) `enet_tx_dirty_ring_t enet_handle_t::txDirtyRing[FSL_FEATURE_ENET_QUEUE]`
- (11) `bool enet_handle_t::txReclaimEnable[FSL_FEATURE_ENET_QUEUE]`
- (12) `enet_rx_alloc_callback_t enet_handle_t::rxBuffAlloc`
- (13) `enet_rx_free_callback_t enet_handle_t::rxBuffFree`

## 20.5 Macro Definition Documentation

20.5.1 **#define FSL\_ENET\_DRIVER\_VERSION (MAKE\_VERSION(2, 7, 0))**

20.5.2 **#define ENET\_BUFFDESCRIPTOR\_RX\_EMPTY\_MASK 0x8000U**

20.5.3 **#define ENET\_BUFFDESCRIPTOR\_RX\_SOFTOWNER1\_MASK 0x4000U**

20.5.4 **#define ENET\_BUFFDESCRIPTOR\_RX\_WRAP\_MASK 0x2000U**

20.5.5 **#define ENET\_BUFFDESCRIPTOR\_RX\_SOFTOWNER2\_Mask 0x1000U**

20.5.6 **#define ENET\_BUFFDESCRIPTOR\_RX\_LAST\_MASK 0x0800U**

20.5.7 **#define ENET\_BUFFDESCRIPTOR\_RX\_MISS\_MASK 0x0100U**

20.5.8 **#define ENET\_BUFFDESCRIPTOR\_RX\_BROADCAST\_MASK 0x0080U**

20.5.9 **#define ENET\_BUFFDESCRIPTOR\_RX\_MULTICAST\_MASK 0x0040U**

20.5.10 **#define ENET\_BUFFDESCRIPTOR\_RX\_LENVIOLATE\_MASK 0x0020U**

20.5.11 **#define ENET\_BUFFDESCRIPTOR\_RX\_NOOCTET\_MASK 0x0010U**

20.5.12 **#define ENET\_BUFFDESCRIPTOR\_RX\_CRC\_MASK 0x0004U**

20.5.13 **#define ENET\_BUFFDESCRIPTOR\_RX\_OVERRUN\_MASK 0x0002U**

20.5.14 **#define ENET\_BUFFDESCRIPTOR\_RX\_TRUNC\_MASK 0x0001U**

20.5.15 **#define ENET\_BUFFDESCRIPTOR\_TX\_READY\_MASK 0x8000U**

20.5.16 **#define ENET\_BUFFDESCRIPTOR\_TX\_SOFTOWENER1\_MASK 0x4000U**

20.5.17 **#define ENET\_BUFFDESCRIPTOR\_TX\_WRAP\_MASK 0x2000U**

20.5.18 **#define ENET\_BUFFDESCRIPTOR\_TX\_SOFTOWENER2\_MASK 0x1000U**

20.5.19 **#define ENET\_BUFFDESCRIPTOR\_TX\_LAST\_MASK 0x0800U**

20.5.20 **#define ENET\_BUFFDESCRIPTOR\_TX\_TRANSMITCRC\_MASK 0x0400U**

20.5.21 **#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)
```

**20.5.22 #define ENET\_FRAME\_MAX\_FRAMELEN 1518U**

**20.5.23 #define ENET\_FRAME\_VLAN\_TAGLEN 4U**

**20.5.24 #define ENET\_FRAME\_CRC\_LEN 4U**

**20.5.25 #define ENET\_FIFO\_MIN\_RX\_FULL 5U**

**20.5.26 #define ENET\_RX\_MIN\_BUFFERSIZE 256U**

**20.5.27 #define ENET\_PHY\_MAXADDRESS (ENET\_MMFR\_PA\_MASK >>  
ENET\_MMFR\_PA\_SHIFT)**

**20.5.28 #define ENET\_TX\_INTERRUPT**

**Value:**

```
((uint32_t)kENET_TxFrameInterrupt | (uint32_t)
 kENET_TxBufferInterrupt | (uint32_t)
 kENET_TxFrame1Interrupt | \
 (uint32_t)kENET_TxBuffer1Interrupt | (uint32_t)
 kENET_TxFrame2Interrupt |
 (uint32_t)kENET_TxBuffer2Interrupt) \
```

**20.5.29 #define ENET\_RX\_INTERRUPT**

**Value:**

```
((uint32_t)kENET_RxFrameInterrupt | (uint32_t)
 kENET_RxBufferInterrupt | (uint32_t)
 kENET_RxFrame1Interrupt | \
 (uint32_t)kENET_RxBuffer1Interrupt | (uint32_t)
 kENET_RxFrame2Interrupt |
 (uint32_t)kENET_RxBuffer2Interrupt) \
```

**20.5.30 #define ENET\_TS\_INTERRUPT ((uint32\_t)kENET\_TsTimerInterrupt | (uint32\_t)kENET\_TsAvailInterrupt)**

**20.5.31 #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)
```

## 20.6 Typedef Documentation

**20.6.1 typedef void>(\* enet\_rx\_alloc\_callback\_t)(ENET\_Type \*base, void \*userData, uint8\_t ringId)**

**20.6.2 typedef void(\* enet\_rx\_free\_callback\_t)(ENET\_Type \*base, void \*buffer, void \*userData, uint8\_t ringId)**

**20.6.3 typedef void(\* enet\_callback\_t)(ENET\_Type \*base, enet\_handle\_t \*handle, uint32\_t ringId, enet\_event\_t event, enet\_frame\_info\_t \*frameInfo, void \*userData)**

**20.6.4 typedef void(\* enet\_isr\_ring\_t)(ENET\_Type \*base, enet\_handle\_t \*handle, uint32\_t ringId)**

## 20.7 Enumeration Type Documentation

### 20.7.1 anonymous enum

Enumerator

***kStatus\_ENET\_InitMemoryFail*** Init fails since buffer memory is not enough.

***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\_RxFrameDrop*** Rx frame is dropped since no buffer memory.

***kStatus\_ENET\_TxFrameOverLen*** Tx frame over length.

***kStatus\_ENET\_TxFrameBusy*** Tx buffer descriptors are under process.

***kStatus\_ENET\_TxFrameFail*** Transmit frame fail.



### 20.7.2 enum enet\_mii\_mode\_t

Enumerator

*kENET\_MiiMode* MII mode for data interface.  
*kENET\_RmiiMode* RMII mode for data interface.  
*kENET\_RgmiiMode* RGMII mode for data interface.

### 20.7.3 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.  
*kENET\_MiiSpeed1000M* Speed 1000M bps.

### 20.7.4 enum enet\_mii\_duplex\_t

Enumerator

*kENET\_MiiHalfDuplex* Half duplex mode.  
*kENET\_MiiFullDuplex* Full duplex mode.

### 20.7.5 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.

### 20.7.6 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.

### 20.7.7 enum enet\_mii\_extend\_opcode

Enumerator

- kENET\_MiiAddrWrite\_C45* Address Write operation.
- kENET\_MiiWriteFrame\_C45* Write frame operation for a valid MII management frame.
- kENET\_MiiReadFrame\_C45* Read frame operation for a valid MII management frame.

### 20.7.8 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).
- kENET\_ControlSVLANEnable* Enable S-VLAN.
- kENET\_ControlVLANUseSecondTag* Enable extracting the second vlan tag for further processing.

### 20.7.9 enum enet\_interrupt\_enable\_t

This enumeration uses one-bit 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.
- 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\_RxFlush2Interrupt*** Rx DMA ring2 flush indication.  
***kENET\_RxFlush1Interrupt*** Rx DMA ring1 flush indication.  
***kENET\_RxFlush0Interrupt*** RX DMA ring0 flush indication.  
***kENET\_TxFrame2Interrupt*** Tx frame interrupt for Tx ring/class 2.  
***kENET\_TxBuffer2Interrupt*** Tx buffer interrupt for Tx ring/class 2.  
***kENET\_RxFrame2Interrupt*** Rx frame interrupt for Rx ring/class 2.  
***kENET\_RxBuffer2Interrupt*** Rx buffer interrupt for Rx ring/class 2.  
***kENET\_TxFrame1Interrupt*** Tx frame interrupt for Tx ring/class 1.  
***kENET\_TxBuffer1Interrupt*** Tx buffer interrupt for Tx ring/class 1.  
***kENET\_RxFrame1Interrupt*** Rx frame interrupt for Rx ring/class 1.  
***kENET\_RxBuffer1Interrupt*** Rx buffer interrupt for Rx ring/class 1.  
***kENET\_TsAvailInterrupt*** TS AVAIL interrupt source for PTP.  
***kENET\_TsTimerInterrupt*** TS WRAP interrupt source for PTP.

## 20.7.10 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.

## 20.7.11 enum enet\_idle\_slope\_t

Enumerator

***kENET\_IdleSlope1*** The bandwidth fraction is about 0.002.  
***kENET\_IdleSlope2*** The bandwidth fraction is about 0.003.  
***kENET\_IdleSlope4*** The bandwidth fraction is about 0.008.  
***kENET\_IdleSlope8*** The bandwidth fraction is about 0.02.  
***kENET\_IdleSlope16*** The bandwidth fraction is about 0.03.

*kENET\_IdleSlope32* The bandwidth fraction is about 0.06.  
*kENET\_IdleSlope64* The bandwidth fraction is about 0.11.  
*kENET\_IdleSlope128* The bandwidth fraction is about 0.20.  
*kENET\_IdleSlope256* The bandwidth fraction is about 0.33.  
*kENET\_IdleSlope384* The bandwidth fraction is about 0.43.  
*kENET\_IdleSlope512* The bandwidth fraction is about 0.50.  
*kENET\_IdleSlope640* The bandwidth fraction is about 0.56.  
*kENET\_IdleSlope768* The bandwidth fraction is about 0.60.  
*kENET\_IdleSlope896* The bandwidth fraction is about 0.64.  
*kENET\_IdleSlope1024* The bandwidth fraction is about 0.67.  
*kENET\_IdleSlope1152* The bandwidth fraction is about 0.69.  
*kENET\_IdleSlope1280* The bandwidth fraction is about 0.71.  
*kENET\_IdleSlope1408* The bandwidth fraction is about 0.73.  
*kENET\_IdleSlope1536* The bandwidth fraction is about 0.75.

## 20.7.12 enum enet\_tx\_accelerator\_t

Enumerator

*kENET\_TxAccelIsShift16Enabled* Transmit FIFO shift-16.  
*kENET\_TxAccelIpChecksumEnabled* Insert IP header checksum.  
*kENET\_TxAccelProtoChecksumEnabled* Insert protocol checksum.

## 20.7.13 enum enet\_rx\_accelerator\_t

Enumerator

*kENET\_RxAccelPadRemoveEnabled* Padding removal for short IP frames.  
*kENET\_RxAccelIpChecksumEnabled* Discard with wrong IP header checksum.  
*kENET\_RxAccelProtoChecksumEnabled* Discard with wrong protocol checksum.  
*kENET\_RxAccelMacChecksumEnabled* Discard with Mac layer errors.  
*kENET\_RxAccelIsShift16Enabled* Receive FIFO shift-16.

## 20.8 Function Documentation

### 20.8.1 uint32\_t ENET\_GetInstance ( ENET\_Type \* *base* )

## Parameters

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

## Returns

ENET instance.

### 20.8.2 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. |
|---------------|----------------------------------------------------------|

### 20.8.3 status\_t 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\(\)](#).

## 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

## Return values

|                                     |                                               |
|-------------------------------------|-----------------------------------------------|
| <i>kStatus_Success</i>              | Succeed to initialize the ethernet driver.    |
| <i>kStatus_ENET_Init-MemoryFail</i> | Init fails since buffer memory is not enough. |

#### 20.8.4 **status\_t 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

Return values

|                                     |                                               |
|-------------------------------------|-----------------------------------------------|
| <i>kStatus_Success</i>              | Succeed to initialize the ethernet driver.    |
| <i>kStatus_ENET_Init-MemoryFail</i> | Init fails since buffer memory is not enough. |

### 20.8.5 void ENET\_Down ( ENET\_Type \* *base* )

This function disables the ENET module.

Parameters

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

### 20.8.6 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. |
|-------------|-------------------------------|

**20.8.7 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.



Parameters

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

### 20.8.8 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 dulpex 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.  |

### 20.8.9 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. See clock distribution.                                                                                  |
| <i>isPreamble-Disabled</i> | The preamble disable flag. <ul style="list-style-type: none"> <li>• true Enables the preamble.</li> <li>• false Disables the preamble.</li> </ul> |

### 20.8.10 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.

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

Parameters

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

Returns

The data read from PHY

**20.8.12** `static void ENET_StartSMIWrite ( ENET_Type * base, uint8_t phyAddr, uint8_t regAddr, enet_mii_write_t operation, uint16_t data ) [inline], [static]`

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated [ENET\\_MDIOWrite\(\)](#) can be called. For customized requirements, implement with combining separated APIs.

Parameters

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

**20.8.13** `static void ENET_StartSMIRead ( ENET_Type * base, uint8_t phyAddr, uint8_t regAddr, enet_mii_read_t operation ) [inline], [static]`

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated [ENET\\_MDIORead\(\)](#) can be called. For customized requirements, implement with combining separated APIs.

## Parameters

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

### 20.8.14 `status_t ENET_MDIOWrite ( ENET_Type * base, uint8_t phyAddr, uint8_t regAddr, uint16_t data )`

## Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | ENET peripheral base address.        |
| <i>phyAddr</i> | The PHY address. Range from 0 ~ 31.  |
| <i>regAddr</i> | The PHY register. Range from 0 ~ 31. |
| <i>data</i>    | The data written to PHY.             |

## Returns

kStatus\_Success MDIO access succeeds.  
kStatus\_Timeout MDIO access timeout.

### 20.8.15 `status_t ENET_MDIORead ( ENET_Type * base, uint8_t phyAddr, uint8_t regAddr, uint16_t * pData )`

## Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | ENET peripheral base address.        |
| <i>phyAddr</i> | The PHY address. Range from 0 ~ 31.  |
| <i>regAddr</i> | The PHY register. Range from 0 ~ 31. |
| <i>pData</i>   | The data read from PHY.              |

## Returns

kStatus\_Success MDIO access succeeds.  
kStatus\_Timeout MDIO access timeout.

### 20.8.16 **static void ENET\_StartExtC45SMIWriteReg ( ENET\_Type \* *base*, uint8\_t *portAddr*, uint8\_t *devAddr*, uint16\_t *regAddr* ) [inline], [static]**

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated [ENET\\_MDIO\\_C45\\_Write\(\)](#)/[ENET\\_MDIO\\_C45\\_Read\(\)](#) can be called. For customized requirements, implement with combining separated APIs.

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>base</i>     | ENET peripheral base address.       |
| <i>portAddr</i> | The MDIO port address(PHY address). |
| <i>devAddr</i>  | The device address.                 |
| <i>regAddr</i>  | The PHY register address.           |

### 20.8.17 **static void ENET\_StartExtC45SMIWriteData ( ENET\_Type \* *base*, uint8\_t *portAddr*, uint8\_t *devAddr*, uint16\_t *data* ) [inline], [static]**

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated [ENET\\_MDIO\\_C45\\_Write\(\)](#) can be called. For customized requirements, implement with combining separated APIs.

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>base</i>     | ENET peripheral base address.       |
| <i>portAddr</i> | The MDIO port address(PHY address). |
| <i>devAddr</i>  | The device address.                 |
| <i>data</i>     | The data written to PHY.            |

### 20.8.18 **static void ENET\_StartExtC45SMIReadData ( ENET\_Type \* *base*, uint8\_t *portAddr*, uint8\_t *devAddr* ) [inline], [static]**

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated [ENET\\_MDIO\\_C45\\_Read\(\)](#) can be called. For customized requirements, implement with combining separated APIs.

## Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>base</i>     | ENET peripheral base address.       |
| <i>portAddr</i> | The MDIO port address(PHY address). |
| <i>devAddr</i>  | The device address.                 |

### 20.8.19 `status_t ENET_MDIOC45Write ( ENET_Type * base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t data )`

## Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>base</i>     | ENET peripheral base address.       |
| <i>portAddr</i> | The MDIO port address(PHY address). |
| <i>devAddr</i>  | The device address.                 |
| <i>regAddr</i>  | The PHY register address.           |
| <i>data</i>     | The data written to PHY.            |

## Returns

kStatus\_Success MDIO access succeeds.  
kStatus\_Timeout MDIO access timeout.

### 20.8.20 `status_t ENET_MDIOC45Read ( ENET_Type * base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t * pData )`

## Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>base</i>     | ENET peripheral base address.       |
| <i>portAddr</i> | The MDIO port address(PHY address). |
| <i>devAddr</i>  | The device address.                 |
| <i>regAddr</i>  | The PHY register address.           |

|              |                         |
|--------------|-------------------------|
| <i>pData</i> | The data read from PHY. |
|--------------|-------------------------|

## Returns

kStatus\_Success MDIO access succeeds.  
kStatus\_Timeout MDIO access timeout.

### 20.8.21 static void ENET\_SetRGMIIIClockDelay ( ENET\_Type \* *base*, bool *txEnabled*, bool *rxEnabled* ) [inline], [static]

## Parameters

|                  |                                                                 |
|------------------|-----------------------------------------------------------------|
| <i>base</i>      | ENET peripheral base address.                                   |
| <i>txEnabled</i> | Enable or disable to generate the delayed version of RGMII_TXC. |
| <i>rxEnabled</i> | Enable or disable to use the delayed version of RGMII_RXC.      |

### 20.8.22 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. |

### 20.8.23 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. |

### 20.8.24 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. |

### 20.8.25 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. |

### 20.8.26 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\(\)](#). This should be called when the frame reception is required.

Parameters

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

### 20.8.27 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. |

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



## Parameters

|                      |                                                                                                                                                |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| <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.  |

### 20.8.29 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 <a href="#">enet_interrupt_enable_t</a> . |

### 20.8.30 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 <a href="#">enet_interrupt_enable_t</a> . |

### 20.8.31 `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](#).

### 20.8.32 `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 <a href="#">enet_interrupt_enable_t</a> . |

### 20.8.33 `void ENET_SetRxISRHandler ( ENET_Type * base, enet_isr_ring_t` `ISRHandler )`

## Parameters

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

### 20.8.34 void ENET\_SetTxISRHandler ( ENET\_Type \* *base*, enet\_isr\_ring\_t *ISRHandler* )

## Parameters

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

### 20.8.35 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.       |

### 20.8.36 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_INSTANCE_QUEUEEn(x) - 1).                  |

### 20.8.37 void ENET\_GetStatistics ( ENET\_Type \* *base*, enet\_transfer\_stats\_t \* *statistics* )

## Parameters

|                   |                                   |
|-------------------|-----------------------------------|
| <i>base</i>       | ENET peripheral base address.     |
| <i>statistics</i> | The statistics structure pointer. |

### 20.8.38 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 receive frame and update the BD 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.                                                                     |
| <i>kStatus_ENET_RxFrame-Error</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. |

**20.8.39 status\_t ENET\_ReadFrame ( ENET\_Type \* base, enet\_handle\_t \* handle, uint8\_t \* data, uint32\_t length, uint8\_t ringId, 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

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 API uses memcpy to copy data from DMA buffer to application buffer, 4 bytes aligned data buffer in 32 bits platforms provided by user may let compiler use optimization instruction to reduce time consumption. 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.

**20.8.40** `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. This API uses memcpy to copy data from DMA buffer to application buffer, 4 bytes aligned data buffer in 32 bits platforms provided by user may let compiler use optimization instruction to reduce time consumption.

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. |

**20.8.41 status\_t ENET\_SetTxReclaim ( enet\_handle\_t \* handle, bool isEnabled, uint8\_t ringId )**

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.    |

#### 20.8.42 void ENET\_ReclaimTxDescriptor ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, uint8\_t *ringId* )

This function is used to update the tx descriptor status and store the tx timestamp when the 1588 feature is enabled. This is called by the transmit interrupt IRQ handler after the complete of a frame transmission.

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>ringId</i> | The ring index or ring number.                                                    |

#### 20.8.43 status\_t ENET\_GetRxFrame ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, enet\_rx\_frame\_struct\_t \* *rxFrame*, uint8\_t *ringId* )

This function uses the user-defined allocation and free callbacks. Every time application gets one frame through this function, driver stores the buffer address(es) in enet\_buffer\_struct\_t and allocate new buffer(s) for the BD(s). If there's no memory buffer in the pool, this function drops current one frame to keep the Rx frame in BD ring is as fresh as possible.

Note

Application must provide a memory pool including at least BD number + n buffers in order for this function to work properly, because each BD must always take one buffer while driver is running, then other extra n buffer(s) can be taken by application. Here n is the ceil(max\_frame\_length(set by RCR) / bd\_rx\_size(set by MRBR)). Application must also provide an array structure in rxFrame->rxBuffArray with n index to receive one complete frame in any case.



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>rxFrame</i> | The received frame information structure provided by user.                        |
| <i>ringId</i>  | The ring index or ring number.                                                    |

Return values

|                                   |                                                                 |
|-----------------------------------|-----------------------------------------------------------------|
| <i>kStatus_Success</i>            | Succeed to get one frame and allocate new memory for Rx buffer. |
| <i>kStatus_ENET_RxFrame-Empty</i> | There's no Rx frame in the BD.                                  |
| <i>kStatus_ENET_RxFrame-Error</i> | There's issue in this receiving.                                |
| <i>kStatus_ENET_RxFrame-Drop</i>  | There's no new buffer memory for BD, drop this frame.           |

**20.8.44 status\_t ENET\_StartTxFrame ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, enet\_tx\_frame\_struct\_t \* *txFrame*, uint8\_t *ringId* )**

This function supports scattered buffer transmit, user needs to provide the buffer array.

Note

Tx reclaim should be enabled to ensure the Tx buffer ownership can be given back to application after Tx is over.

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>txFrame</i> | The Tx frame structure.                                                           |
| <i>ringId</i>  | The ring index or ring number.                                                    |

Return values

---

|                                     |                                                                         |
|-------------------------------------|-------------------------------------------------------------------------|
| <i>kStatus_Success</i>              | Succeed to send one frame.                                              |
| <i>kStatus_ENET_TxFrame-Busy</i>    | The BD is not ready for Tx or the reclaim operation still not finishes. |
| <i>kStatus_ENET_TxFrame-OverLen</i> | The Tx frame length is over max ethernet frame length.                  |

**20.8.45 void ENET\_TransmitIRQHandler ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, uint32\_t *ringId* )**

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | ENET peripheral base address. |
| <i>handle</i> | The ENET handler pointer.     |
| <i>ringId</i> | The ring id or ring number.   |

**20.8.46 void ENET\_ReceiveIRQHandler ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, uint32\_t *ringId* )**

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | ENET peripheral base address. |
| <i>handle</i> | The ENET handler pointer.     |
| <i>ringId</i> | The ring id or ring number.   |

**20.8.47 void ENET\_CommonFrame1IRQHandler ( ENET\_Type \* *base* )**

This is used for the combined tx/rx interrupt for multi-ring (frame 1).

Parameters

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

**20.8.48 void ENET\_CommonFrame2IRQHandler ( ENET\_Type \* *base* )**

This is used for the combined tx/rx interrupt for multi-ring (frame 2).

Parameters

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

**20.8.49 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.     |

**20.8.50 void ENET\_Ptp1588IRQHandler ( ENET\_Type \* *base* )**

This is used for the 1588 timer interrupt.

Parameters

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

**20.8.51 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. |
|-------------|-------------------------------|

## 20.9 Variable Documentation

**20.9.1 const clock\_ip\_name\_t s\_enetClock[]**

## 20.10 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.

### 20.10.1 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);
```

```

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 21

# EWM: External Watchdog Monitor Driver

### 21.1 Overview

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

### 21.2 Typical use case

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

### Data Structures

- struct `ewm_config_t`  
*Describes EWM clock source. [More...](#)*

### Enumerations

- enum `_ewm_interrupt_enable_t` { `kEWM_InterruptEnable` = `EWM_CTRL_INTEN_MASK` }  
*EWM interrupt configuration structure with default settings all disabled.*
- enum `_ewm_status_flags_t` { `kEWM_RunningFlag` = `EWM_CTRL_EWMEN_MASK` }  
*EWM status flags.*

### Driver version

- `#define FSL_EWM_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))`  
*EWM driver version 2.0.3.*

### EWM initialization and de-initialization

- void `EWM_Init` (`EWM_Type *base`, const `ewm_config_t *config`)  
*Initializes the EWM peripheral.*
- void `EWM_Deinit` (`EWM_Type *base`)  
*Deinitializes the EWM peripheral.*
- void `EWM_GetDefaultConfig` (`ewm_config_t *config`)  
*Initializes the EWM configuration structure.*

### EWM functional Operation

- static void `EWM_EnableInterrupts` (`EWM_Type *base`, `uint32_t mask`)  
*Enables the EWM interrupt.*
- static void `EWM_DisableInterrupts` (`EWM_Type *base`, `uint32_t mask`)  
*Disables the EWM interrupt.*
- static `uint32_t EWM_GetStatusFlags` (`EWM_Type *base`)  
*Gets all status flags.*

- void [EWM\\_Refresh](#) (EWM\_Type \*base)  
*Services the EWM.*

### 21.3 Data Structure Documentation

#### 21.3.1 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.*

### 21.4 Macro Definition Documentation

#### 21.4.1 #define FSL\_EWM\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 3))

### 21.5 Enumeration Type Documentation

#### 21.5.1 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.

#### 21.5.2 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.

## 21.6 Function Documentation

### 21.6.1 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 |

### 21.6.2 void EWM\_Deinit ( EWM\_Type \* *base* )

This function is used to shut down the EWM.

#### Parameters

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

### 21.6.3 void EWM\_GetDefaultConfig ( ewm\_config\_t \* *config* )

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

```
* ewmConfig->enableEwm = true;
* ewmConfig->enableEwmInput = false;
* ewmConfig->setInputAssertLogic = false;
* ewmConfig->enableInterrupt = false;
* ewmConfig->ewm_lpo_clock_source_t = kEWM_LpoClockSource0;
* ewmConfig->prescaler = 0;
* ewmConfig->compareLowValue = 0;
* ewmConfig->compareHighValue = 0xFEU;
*
```



Parameters

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

See Also

[ewm\\_config\\_t](#)

#### 21.6.4 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"> <li>• kEWM_InterruptEnable</li> </ul> |

#### 21.6.5 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"> <li>• kEWM_InterruptEnable</li> </ul> |

#### 21.6.6 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.

### 21.6.7 void EWM\_Refresh ( EWM\_Type \* *base* )

This function resets the EWM counter to zero.

Parameters

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



## Chapter 22

# FlexCAN: Flex Controller Area Network Driver

### 22.1 Overview

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

### Modules

- [FlexCAN Driver](#)

## 22.2 FlexCAN Driver

### 22.2.1 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.

### 22.2.2 Typical use case

#### 22.2.2.1 Message Buffer Send Operation

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

#### 22.2.2.2 Message Buffer Receive Operation

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

#### 22.2.2.3 Receive FIFO Operation

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

## Data Structures

- struct [flexcan\\_memory\\_error\\_report\\_status\\_t](#)  
*FlexCAN memory error register status structure. [More...](#)*
- 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 Legacy 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 length 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 Legacy 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.*  
 • #define `FLEXCAN_MECR_INT_MASK(x) (((uint64_t)(((uint64_t)(x)) << 16)) & 0xD0000000-U)`
- *FlexCAN interrupt/status flag helper macro.*  
 • #define `FLEXCAN_MEMORY_ENHANCED_RX_FIFO_INIT_FLAG (0U)`
- *FlexCAN Enhanced Rx FIFO base address helper macro.*  
 • #define `FLEXCAN_CALLBACK(x) void(x)(CAN_Type * base, flexcan_handle_t * handle, status-_t status, uint64_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\_RxFifoDisabled,
  - kStatus\_FLEXCAN\_ErrorStatus = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 11),
  - kStatus\_FLEXCAN\_WakeUp = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 12),
  - kStatus\_FLEXCAN\_UnHandled = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 13),
  - kStatus\_FLEXCAN\_RxRemote = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 14) }

*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 Enhanced/Legacy Rx FIFO priority.*

- enum `_flexcan_interrupt_enable` {
  - `kFLEXCAN_BusOffInterruptEnable` = CAN\_CTRL1\_BOFFMSK\_MASK,
  - `kFLEXCAN_ErrorInterruptEnable` = CAN\_CTRL1\_ERRMSK\_MASK,
  - `kFLEXCAN_TxWarningInterruptEnable` = CAN\_CTRL1\_TWRNMSK\_MASK,
  - `kFLEXCAN_RxWarningInterruptEnable` = CAN\_CTRL1\_RWRNMSK\_MASK,
  - `kFLEXCAN_WakeUpInterruptEnable` = CAN\_MCR\_WAKMSK\_MASK,
  - `kFLEXCAN_HostAccessNCErrInterruptEnable` = FLEXCAN\_MECR\_INT\_MASK(CAN\_MECR\_HANCEI\_MSK\_MASK),
  - `kFLEXCAN_FlexCanAccessNCErrInterruptEnable` = FLEXCAN\_MECR\_INT\_MASK(CAN\_MECR\_FANCEI\_MSK\_MASK),
  - `kFLEXCAN_HostOrFlexCanCErrInterruptEnable` = FLEXCAN\_MECR\_INT\_MASK(CAN\_MECR\_CEI\_MSK\_MASK) }

*FlexCAN interrupt enable enumerations.*

- enum `_flexcan_flags` {
  - `kFLEXCAN_SynchFlag` = CAN\_ESR1\_SYNCH\_MASK,
  - `kFLEXCAN_TxWarningIntFlag` = CAN\_ESR1\_TWRNINT\_MASK,
  - `kFLEXCAN_RxWarningIntFlag` = CAN\_ESR1\_RWRNINT\_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 ,
  - `kFLEXCAN_HostAccessNonCorrectableErrorIntFlag` = FLEXCAN\_MECR\_INT\_MASK(CAN\_ERRSR\_HANCEIF\_MASK),
  - `kFLEXCAN_FlexCanAccessNonCorrectableErrorIntFlag` = FLEXCAN\_MECR\_INT\_MASK(CAN\_ERRSR\_FANCEIF\_MASK),
  - `kFLEXCAN_CorrectableErrorIntFlag` = FLEXCAN\_MECR\_INT\_MASK(CAN\_ERRSR\_CEIF\_MASK),
  - `kFLEXCAN_HostAccessNonCorrectableErrorOverrunFlag` = FLEXCAN\_MECR\_INT\_MASK(CAN\_ERRSR\_HANCEIOF\_MASK),
  - `kFLEXCAN_FlexCanAccessNonCorrectableErrorOverrunFlag` = FLEXCAN\_MECR\_INT\_MASK(CAN\_ERRSR\_FANCEIOF\_MASK),
  - `kFLEXCAN_CorrectableErrorOverrunFlag` = FLEXCAN\_MECR\_INT\_MASK(CAN\_ERRSR\_CEIFIOF\_MASK),
  - `kFLEXCAN_AllMemoryErrorFlag` }

*FlexCAN status flags.*

- enum `_flexcan_error_flags` {



```

kFLEXCAN_TxErrorWarningFlag = CAN_ESR1_TXWRN_MASK,
kFLEXCAN_RxErrorWarningFlag = CAN_ESR1_RXWRN_MASK,
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 Legacy Rx FIFO status flags.*

- enum flexcan\_memory\_error\_type\_t {
 

```

kFLEXCAN_CorrectableError = 0U,
kFLEXCAN_NonCorrectableError }

```

*FlexCAN Memory Error Type.*

- enum flexcan\_memory\_access\_type\_t {
 

```

kFLEXCAN_MoveOutFlexCanAccess = 0U,
kFLEXCAN_MoveInAccess,
kFLEXCAN_TxArbitrationAccess,
kFLEXCAN_RxMatchingAccess,
kFLEXCAN_MoveOutHostAccess }

```

*FlexCAN Memory Access Type.*

- enum flexcan\_byte\_error\_syndrome\_t {
 

```

kFLEXCAN_NoError = 0U,
kFLEXCAN_ParityBits0Error = 1U,
kFLEXCAN_ParityBits1Error = 2U,
kFLEXCAN_ParityBits2Error = 4U,
kFLEXCAN_ParityBits3Error = 8U,
kFLEXCAN_ParityBits4Error = 16U,
kFLEXCAN_DataBits0Error = 28U,
kFLEXCAN_DataBits1Error = 22U,
kFLEXCAN_DataBits2Error = 19U,
kFLEXCAN_DataBits3Error = 25U,
kFLEXCAN_DataBits4Error = 26U,
kFLEXCAN_DataBits5Error = 7U,
kFLEXCAN_DataBits6Error = 21U,
kFLEXCAN_DataBits7Error = 14U,
kFLEXCAN_AllZeroError = 6U,
kFLEXCAN_AllOneError = 31U,
kFLEXCAN_NonCorrectableErrors }

```

*FlexCAN Memory Error Byte Syndrome.*

## Driver version

- #define `FSL_FLEXCAN_DRIVER_VERSION` (`MAKE_VERSION(2, 11, 1)`)  
*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` (`CAN_Type *base`, uint32\_t bitRate, uint32\_t sourceClock\_Hz, `flexcan_timing_config_t *pTimingConfig`)  
*Calculates the improved timing values by specific bit Rates 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 classical CAN protocol timing characteristic.*
- `status_t` `FLEXCAN_SetBitRate` (`CAN_Type *base`, uint32\_t sourceClock\_Hz, uint32\_t bitRate\_Bps)  
*Set bit rate of FlexCAN classical CAN frame or CAN FD frame nominal phase.*
- 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 Legacy Rx FIFO.*

## Status

- static uint64\_t [FLEXCAN\\_GetStatusFlags](#) (CAN\_Type \*base)  
*Gets the FlexCAN module interrupt flags.*
- static void [FLEXCAN\\_ClearStatusFlags](#) (CAN\_Type \*base, uint64\_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 uint64\_t [FLEXCAN\\_GetMbStatusFlags](#) (CAN\_Type \*base, uint64\_t mask)  
*Gets the FlexCAN Message Buffer interrupt flags.*
- static void [FLEXCAN\\_ClearMbStatusFlags](#) (CAN\_Type \*base, uint64\_t mask)  
*Clears the FlexCAN Message Buffer interrupt flags.*
- void [FLEXCAN\\_GetMemoryErrorReportStatus](#) (CAN\_Type \*base, flexcan\_memory\_error\_report-\_status\_t \*errorStatus)  
*Gets the FlexCAN Memory Error Report registers status.*

## Interrupts

- static void [FLEXCAN\\_EnableInterrupts](#) (CAN\_Type \*base, uint64\_t mask)  
*Enables FlexCAN interrupts according to the provided mask.*
- static void [FLEXCAN\\_DisableInterrupts](#) (CAN\_Type \*base, uint64\_t mask)  
*Disables FlexCAN interrupts according to the provided mask.*
- static void [FLEXCAN\\_EnableMbInterrupts](#) (CAN\_Type \*base, uint64\_t mask)  
*Enables FlexCAN Message Buffer interrupts.*
- static void [FLEXCAN\\_DisableMbInterrupts](#) (CAN\_Type \*base, uint64\_t mask)  
*Disables FlexCAN Message Buffer interrupts.*

## DMA Control

- void [FLEXCAN\\_EnableRxFifoDMA](#) (CAN\_Type \*base, bool enable)  
*Enables or disables the FlexCAN Rx FIFO DMA request.*
- static uintptr\_t [FLEXCAN\\_GetRxFifoHeadAddr](#) (CAN\_Type \*base)  
*Gets the Rx FIFO Head address.*

## 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\\_ReadRxFifo](#) (CAN\_Type \*base, flexcan\_frame\_t \*pRxFrame)  
*Reads a FlexCAN Message from Legacy 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 Legacy 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.*
- [status\\_t FLEXCAN\\_TransferGetReceiveFifoCount](#) (CAN\_Type \*base, [flexcan\\_handle\\_t](#) \*handle, [size\\_t](#) \*count)  
*Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.*
- [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.*

## 22.2.3 Data Structure Documentation

### 22.2.3.1 struct flexcan\_memory\_error\_report\_status\_t

This structure contains the memory access properties that caused a memory error access. It is used as the parameter of [FLEXCAN\\_GetMemoryErrorReportStatus\(\)](#) function. And user can use [FLEXCAN\\_GetMemoryErrorReportStatus](#) to get the status of the last memory error access.

## Data Fields

- `flexcan_memory_error_type_t errorType`  
*The type of memory error that giving rise to the report.*
- `flexcan_memory_access_type_t accessType`  
*The type of memory access that giving rise to the memory error.*
- `uint16_t accessAddress`  
*The address where memory error detected.*
- `uint32_t errorData`  
*The raw data word read from memory with error.*
- `bool byteIsRead`  
*The byte n (0~3) was read or not.*

## Field Documentation

- (1) `flexcan_memory_error_type_t flexcan_memory_error_report_status_t::errorType`
- (2) `flexcan_memory_access_type_t flexcan_memory_error_report_status_t::accessType`
- (3) `uint16_t flexcan_memory_error_report_status_t::accessAddress`
- (4) `uint32_t flexcan_memory_error_report_status_t::errorData`
- (5) `bool flexcan_memory_error_report_status_t::byteIsRead`

The type of error and which bit in byte (n) is affected by the error.

### 22.2.3.2 struct flexcan\_frame\_t

#### Field Documentation

- (1) uint32\_t flexcan\_frame\_t::timestamp
- (2) uint32\_t flexcan\_frame\_t::length
- (3) uint32\_t flexcan\_frame\_t::type
- (4) uint32\_t flexcan\_frame\_t::format
- (5) uint32\_t flexcan\_frame\_t::\_\_pad0\_\_
- (6) uint32\_t flexcan\_frame\_t::idhit
- (7) uint32\_t flexcan\_frame\_t::id
- (8) uint32\_t flexcan\_frame\_t::dataWord0
- (9) uint32\_t flexcan\_frame\_t::dataWord1
- (10) uint8\_t flexcan\_frame\_t::dataByte3
- (11) uint8\_t flexcan\_frame\_t::dataByte2
- (12) uint8\_t flexcan\_frame\_t::dataByte1
- (13) uint8\_t flexcan\_frame\_t::dataByte0
- (14) uint8\_t flexcan\_frame\_t::dataByte7
- (15) uint8\_t flexcan\_frame\_t::dataByte6
- (16) uint8\_t flexcan\_frame\_t::dataByte5
- (17) uint8\_t flexcan\_frame\_t::dataByte4

### 22.2.3.3 struct flexcan\_timing\_config\_t

#### Data Fields

- uint16\_t [preDivider](#)  
*Classic CAN or CAN FD nominal phase bit rate prescaler.*
- uint8\_t [rJumpwidth](#)  
*Classic CAN or CAN FD nominal phase Re-sync Jump Width.*
- uint8\_t [phaseSeg1](#)  
*Classic CAN or CAN FD nominal phase Segment 1.*
- uint8\_t [phaseSeg2](#)  
*Classic CAN or CAN FD nominal phase Segment 2.*
- uint8\_t [propSeg](#)

*Classic CAN or CAN FD nominal phase Propagation Segment.*

### Field Documentation

- (1) `uint16_t flexcan_timing_config_t::preDivider`
- (2) `uint8_t flexcan_timing_config_t::rJumpwidth`
- (3) `uint8_t flexcan_timing_config_t::phaseSeg1`
- (4) `uint8_t flexcan_timing_config_t::phaseSeg2`
- (5) `uint8_t flexcan_timing_config_t::propSeg`

### 22.2.3.4 struct flexcan\_config\_t

**Deprecated** Do not use the `baudRate`. It has been superceded `bitRate`

Do not use the `baudRateFD`. It has been superceded `bitRateFD`

### Data Fields

- `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 and Queue feature.*
- `bool disableSelfReception`  
*Enable or Disable Self Reflection.*
- `bool enableListenOnlyMode`  
*Enable or Disable Listen Only Mode.*
- `bool enableSupervisorMode`  
*Enable or Disable Supervisor Mode, enable this mode will make registers allow only Supervisor access.*
- `bool enableDoze`  
*Enable or Disable Doze Mode.*
- `bool enableMemoryErrorControl`  
*Enable or Disable the memory errors detection and correction mechanism.*
- `bool enableNonCorrectableErrorEnterFreeze`  
*Enable or Disable Non-Correctable Errors In FlexCAN Access Put Device In Freeze Mode.*
- `uint32_t baudRate`  
*FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.*
- `uint32_t bitRate`

*FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.*

### Field Documentation

- (1) `uint32_t flexcan_config_t::baudRate`
- (2) `uint32_t flexcan_config_t::bitRate`
- (3) `flexcan_clock_source_t flexcan_config_t::clkSrc`
- (4) `flexcan_wake_up_source_t flexcan_config_t::wakeupSrc`
- (5) `uint8_t flexcan_config_t::maxMbNum`
- (6) `bool flexcan_config_t::enableLoopBack`
- (7) `bool flexcan_config_t::enableTimerSync`
- (8) `bool flexcan_config_t::enableSelfWakeup`
- (9) `bool flexcan_config_t::enableIndividMask`
- (10) `bool flexcan_config_t::disableSelfReception`
- (11) `bool flexcan_config_t::enableListenOnlyMode`
- (12) `bool flexcan_config_t::enableSupervisorMode`
- (13) `bool flexcan_config_t::enableDoze`
- (14) `bool flexcan_config_t::enableMemoryErrorControl`
- (15) `bool flexcan_config_t::enableNonCorrectableErrorEnterFreeze`

#### 22.2.3.5 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).*



**Field Documentation**

- (1) `uint32_t flexcan_rx_mb_config_t::id`
- (2) `flexcan_frame_format_t flexcan_rx_mb_config_t::format`
- (3) `flexcan_frame_type_t flexcan_rx_mb_config_t::type`

**22.2.3.6 struct flexcan\_rx\_fifo\_config\_t****Data Fields**

- `uint32_t * idFilterTable`  
*Pointer to the FlexCAN Legacy Rx FIFO identifier filter table.*
- `uint8_t idFilterNum`  
*The FlexCAN Legacy Rx FIFO Filter elements quantity.*
- `flexcan_rx_fifo_filter_type_t idFilterType`  
*The FlexCAN Legacy Rx FIFO Filter type.*
- `flexcan_rx_fifo_priority_t priority`  
*The FlexCAN Legacy Rx FIFO receive priority.*

**Field Documentation**

- (1) `uint32_t* flexcan_rx_fifo_config_t::idFilterTable`
- (2) `uint8_t flexcan_rx_fifo_config_t::idFilterNum`
- (3) `flexcan_rx_fifo_filter_type_t flexcan_rx_fifo_config_t::idFilterType`
- (4) `flexcan_rx_fifo_priority_t flexcan_rx_fifo_config_t::priority`

**22.2.3.7 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.*

**Field Documentation**

- (1) `flexcan_frame_t* flexcan_mb_transfer_t::frame`
- (2) `uint8_t flexcan_mb_transfer_t::mbIdx`

**22.2.3.8 struct flexcan\_fifo\_transfer\_t****Data Fields**

- `flexcan_frame_t * frame`

- `size_t frameNum`  
*Number of CAN Message need to be received from Legacy or Enhanced Rx FIFO.*

### Field Documentation

(1) `flexcan_frame_t* flexcan_fifo_transfer_t::frame`

(2) `size_t flexcan_fifo_transfer_t::frameNum`

### 22.2.3.9 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 CAN data from Message Buffers.*
- `flexcan_frame_t *volatile rxFifoFrameBuf`  
*The buffer for received CAN data from Legacy Rx FIFO.*
- `size_t rxFifoFrameNum`  
*The number of CAN messages remaining to be received from Legacy or Enhanced Rx FIFO.*
- `size_t rxFifoTransferTotalNum`  
*Total CAN Message number need to be received from Legacy or Enhanced 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.*

## Field Documentation

- (1) `flexcan_transfer_callback_t flexcan_handle_t::callback`
- (2) `void* flexcan_handle_t::userData`
- (3) `flexcan_frame_t* volatile flexcan_handle_t::mbFrameBuf[CAN_WORD1_COUNT]`
- (4) `flexcan_frame_t* volatile flexcan_handle_t::rxFifoFrameBuf`
- (5) `size_t flexcan_handle_t::rxFifoFrameNum`
- (6) `size_t flexcan_handle_t::rxFifoTransferTotalNum`
- (7) `volatile uint8_t flexcan_handle_t::mbState[CAN_WORD1_COUNT]`
- (8) `volatile uint8_t flexcan_handle_t::rxFifoState`
- (9) `volatile uint32_t flexcan_handle_t::timestamp[CAN_WORD1_COUNT]`

## 22.2.4 Macro Definition Documentation

**22.2.4.1** `#define FSL_FLEXCAN_DRIVER_VERSION (MAKE_VERSION(2, 11, 1))`

**22.2.4.2** `#define FLEXCAN_ID_STD( id ) (((uint32_t)((uint32_t)(id)) << CAN_ID_STD_SHIFT)) & CAN_ID_STD_MASK)`

FlexCAN Frame ID helper macro. Standard Frame ID helper macro.

**22.2.4.3** `#define FLEXCAN_ID_EXT( id )`

**Value:**

```
(((uint32_t)((uint32_t)(id)) << CAN_ID_EXT_SHIFT)) & \
(CAN_ID_EXT_MASK | CAN_ID_STD_MASK)
```

**22.2.4.4** `#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.

**22.2.4.5 #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)
```

**22.2.4.6 #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.

**22.2.4.7 #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)
```

**22.2.4.8 #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)
```

**22.2.4.9 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_HIGH( *id*  
) (((uint32\_t)(id) & 0x7F8) << 21)****22.2.4.10 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_HIGH( *id*  
) (((uint32\_t)(id) & 0x7F8) << 13)****22.2.4.11 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_LOW( *id*  
) (((uint32\_t)(id) & 0x7F8) << 5)****22.2.4.12 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_LOW( *id*  
) (((uint32\_t)(id) & 0x7F8) >> 3)****22.2.4.13 #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))
```

#### 22.2.4.14 #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))
```

#### 22.2.4.15 #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))
```

#### 22.2.4.16 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_HIGH( *id* ) ((FLEXCAN\_ID\_EXT(id) & 0x1FE00000) <<< 3)

#### 22.2.4.17 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_HIGH( *id* )

##### Value:

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 5)
```

#### 22.2.4.18 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_LOW( *id* )

##### Value:

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 13)
```

#### 22.2.4.19 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_LOW( *id* ) ((FLEXCAN\_ID\_EXT(id) & 0x1FE00000) >>> 21)

#### 22.2.4.20 #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.

**22.2.4.21 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )****Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(\
 id, rtr, ide)
```

**22.2.4.22 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_LOW( *id*, *rtr*, *ide* )****Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(\
 id, rtr, ide)
```

**22.2.4.23 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_HIGH( *id* )****Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(\
 id)
```

**22.2.4.24 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_MID\_HIGH( *id* )****Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(\
 id)
```

**22.2.4.25 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_MID\_LOW( *id* )****Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(\
 id)
```

**22.2.4.26 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_LOW( *id* )****Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(\
 id)
```

**22.2.4.27 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_A( *id*, *rtr*, *ide* ) FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_A(id, rtr, ide)**

**22.2.4.28 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(\
 id, rtr, ide)
```

**22.2.4.29 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_B\_LOW( *id*, *rtr*, *ide* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(\
 id, rtr, ide)
```

**22.2.4.30 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_HIGH( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(\
 id)
```

**22.2.4.31 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_MID\_HIGH( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(\
 id)
```

**22.2.4.32 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_MID\_LOW( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(\
 id)
```

- 22.2.4.33** `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW( id ) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)`
- 22.2.4.34** `#define FLEXCAN_MECR_INT_MASK( x ) (((uint64_t)((uint64_t)(x)) << 16)) & 0xD0000000U)`
- 22.2.4.35** `#define FLEXCAN_MEMORY_ENHANCED_RX_FIFO_INIT_FLAG (0U)`
- 22.2.4.36** `#define FLEXCAN_CALLBACK( x ) void(x)(CAN_Type * base, flexcan_handle_t * handle, status_t status, uint64_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.

## 22.2.5 Enumeration Type Documentation

### 22.2.5.1 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\_RxFifoDisabled* Rx Message FIFO is disabled during reading.
- kStatus\_FLEXCAN\_ErrorStatus* FlexCAN Module Error and Status.
- kStatus\_FLEXCAN\_WakeUp* FlexCAN is waken up from STOP mode.
- kStatus\_FLEXCAN\_UnHandled* UnHadled Interrupt asserted.
- kStatus\_FLEXCAN\_RxRemote* Rx Remote Message Received in Mail box.



### 22.2.5.2 enum flexcan\_frame\_format\_t

Enumerator

- kFLEXCAN\_FrameFormatStandard* Standard frame format attribute.
- kFLEXCAN\_FrameFormatExtend* Extend frame format attribute.

### 22.2.5.3 enum flexcan\_frame\_type\_t

Enumerator

- kFLEXCAN\_FrameTypeData* Data frame type attribute.
- kFLEXCAN\_FrameTypeRemote* Remote frame type attribute.

### 22.2.5.4 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.

### 22.2.5.5 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.

### 22.2.5.6 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.

### 22.2.5.7 enum flexcan\_rx\_fifo\_priority\_t

The matching process starts from the Rx MB(or Enhanced/Legacy Rx FIFO) with higher priority. If no MB(or Enhanced/Legacy Rx FIFO filter) is satisfied, the matching process goes on with the Enhanced/Legacy 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 Enhanced/Legacy Rx FIFO first.

### 22.2.5.8 enum \_flexcan\_interrupt\_enable

This provides constants for the FlexCAN interrupt enable enumerations for use in the FlexCAN functions.

Note

FlexCAN Message Buffers and Legacy Rx FIFO interrupts not included in.

Enumerator

***kFLEXCAN\_BusOffInterruptEnable*** Bus Off interrupt, use bit 15.  
***kFLEXCAN\_ErrorInterruptEnable*** CAN Error interrupt, use bit 14.  
***kFLEXCAN\_TxWarningInterruptEnable*** Tx Warning interrupt, use bit 11.  
***kFLEXCAN\_RxWarningInterruptEnable*** Rx Warning interrupt, use bit 10.  
***kFLEXCAN\_WakeUpInterruptEnable*** Self Wake Up interrupt, use bit 26.  
***kFLEXCAN\_HostAccessNCErrorsInterruptEnable*** Host Access With Non-Correctable Errors interrupt, use high word bit 0.  
***kFLEXCAN\_FlexCanAccessNCErrorsInterruptEnable*** FlexCAN Access With Non-Correctable Errors interrupt, use high word bit 2.  
***kFLEXCAN\_HostOrFlexCanCErrorsInterruptEnable*** Host or FlexCAN Access With Correctable Errors interrupt, use high word bit 3.

### 22.2.5.9 enum \_flexcan\_flags

This provides constants for the FlexCAN status flags for use in the FlexCAN functions.

Note

The CPU read action clears the bits corresponding to the FLEXCAN\_ErrorFlag macro, therefore user need to read status flags 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\_IdleFlag* FlexCAN In IDLE Status.  
*kFLEXCAN\_FaultConfinementFlag* FlexCAN Fault Confinement State.  
*kFLEXCAN\_TransmittingFlag* FlexCAN In Transmission Status.  
*kFLEXCAN\_ReceivingFlag* FlexCAN In Reception Status.  
*kFLEXCAN\_BusOffIntFlag* Bus Off Interrupt Flag.  
*kFLEXCAN\_ErrorIntFlag* CAN Error Interrupt Flag.  
*kFLEXCAN\_WakeUpIntFlag* Self Wake-Up Interrupt Flag.  
*kFLEXCAN\_HostAccessNonCorrectableErrorIntFlag* Host Access With Non-Correctable Error Interrupt Flag.  
*kFLEXCAN\_FlexCanAccessNonCorrectableErrorIntFlag* FlexCAN Access With Non-Correctable Error Interrupt Flag.  
*kFLEXCAN\_CorrectableErrorIntFlag* Correctable Error Interrupt Flag.  
*kFLEXCAN\_HostAccessNonCorrectableErrorOverrunFlag* Host Access With Non-Correctable Error Interrupt Overrun Flag.  
*kFLEXCAN\_FlexCanAccessNonCorrectableErrorOverrunFlag* FlexCAN Access With Non-Correctable Error Interrupt Overrun Flag.  
*kFLEXCAN\_CorrectableErrorOverrunFlag* Correctable Error Interrupt Overrun Flag.  
*kFLEXCAN\_AllMemoryErrorFlag* All Memory Error Flags.

#### 22.2.5.10 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\_TxErrorWarningFlag* Tx Error Warning Status.  
*kFLEXCAN\_RxErrorWarningFlag* Rx Error Warning Status.  
*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.

#### 22.2.5.11 anonymous enum

The FlexCAN Legacy 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.

#### 22.2.5.12 enum flexcan\_memory\_error\_type\_t

Enumerator

***kFLEXCAN\_CorrectableError*** The memory error is correctable which means on bit error.  
***kFLEXCAN\_NonCorrectableError*** The memory error is non-correctable which means two bit errors.

#### 22.2.5.13 enum flexcan\_memory\_access\_type\_t

Enumerator

***kFLEXCAN\_MoveOutFlexCanAccess*** The memory error was detected during move-out FlexCAN access.  
***kFLEXCAN\_MoveInAccess*** The memory error was detected during move-in FlexCAN access.  
***kFLEXCAN\_TxArbitrationAccess*** The memory error was detected during Tx Arbitration FlexCAN access.  
***kFLEXCAN\_RxMatchingAccess*** The memory error was detected during Rx Matching FlexCAN access.  
***kFLEXCAN\_MoveOutHostAccess*** The memory error was detected during Rx Matching Host (CPU) access.

#### 22.2.5.14 enum flexcan\_byte\_error\_syndrome\_t

Enumerator

***kFLEXCAN\_NoError*** No bit error in this byte.  
***kFLEXCAN\_ParityBits0Error*** Parity bit 0 error in this byte.  
***kFLEXCAN\_ParityBits1Error*** Parity bit 1 error in this byte.  
***kFLEXCAN\_ParityBits2Error*** Parity bit 2 error in this byte.  
***kFLEXCAN\_ParityBits3Error*** Parity bit 3 error in this byte.  
***kFLEXCAN\_ParityBits4Error*** Parity bit 4 error in this byte.  
***kFLEXCAN\_DataBits0Error*** Data bit 0 error in this byte.  
***kFLEXCAN\_DataBits1Error*** Data bit 1 error in this byte.  
***kFLEXCAN\_DataBits2Error*** Data bit 2 error in this byte.  
***kFLEXCAN\_DataBits3Error*** Data bit 3 error in this byte.  
***kFLEXCAN\_DataBits4Error*** Data bit 4 error in this byte.  
***kFLEXCAN\_DataBits5Error*** Data bit 5 error in this byte.

***kFLEXCAN\_DataBits6Error*** Data bit 6 error in this byte.  
***kFLEXCAN\_DataBits7Error*** Data bit 7 error in this byte.  
***kFLEXCAN\_AllZeroError*** All-zeros non-correctable error in this byte.  
***kFLEXCAN\_AllOneError*** All-ones non-correctable error in this byte.  
***kFLEXCAN\_NonCorrectableErrors*** Non-correctable error in this byte.

## 22.2.6 Function Documentation

### 22.2.6.1 void FLEXCAN\_EnterFreezeMode ( CAN\_Type \* *base* )

This function makes the FlexCAN work under Freeze Mode.

Parameters

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

### 22.2.6.2 void FLEXCAN\_ExitFreezeMode ( CAN\_Type \* *base* )

This function makes the FlexCAN leave Freeze Mode.

Parameters

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

### 22.2.6.3 uint32\_t FLEXCAN\_GetInstance ( CAN\_Type \* *base* )

Parameters

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

Returns

FlexCAN instance.

### 22.2.6.4 bool FLEXCAN\_CalculateImprovedTimingValues ( CAN\_Type \* *base*, uint32\_t *bitRate*, uint32\_t *sourceClock\_Hz*, flexcan\_timing\_config\_t \* *pTimingConfig* )

This function use to calculates the Classical CAN timing values according to the given bit rate. The Calculated timing values will be set in CTRL1/CBT/ENCBT register. The calculation is based on the recommendation of the CiA 301 v4.2.0 and previous version document.

## Parameters

|                       |                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------|
| <i>base</i>           | FlexCAN peripheral base address.                                                        |
| <i>bitRate</i>        | The classical CAN speed in bps defined by user, should be less than or equal to 1-Mbps. |
| <i>sourceClock_Hz</i> | The Source clock frequency in Hz.                                                       |
| <i>pTimingConfig</i>  | Pointer to the FlexCAN timing configuration structure.                                  |

## Returns

TRUE if timing configuration found, FALSE if failed to find configuration.

### 22.2.6.5 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.bitRate = 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, 4000000U);
*
```

## Parameters

|                       |                                                       |
|-----------------------|-------------------------------------------------------|
| <i>base</i>           | FlexCAN peripheral base address.                      |
| <i>pConfig</i>        | Pointer to the user-defined configuration structure.  |
| <i>sourceClock_Hz</i> | FlexCAN Protocol Engine clock source frequency in Hz. |

### 22.2.6.6 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. |
|-------------|----------------------------------|

**22.2.6.7 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->bitRate = 1000000U; flexcanConfig->bitRateFD = 2000000U; flexcanConfig->maxMbNum = 16; flexcanConfig->enableLoopBack = false; flexcanConfig->enableSelfWakeup = false; flexcanConfig->enableIndividMask = false; flexcanConfig->disableSelfReception = false; flexcanConfig->enableListenOnlyMode = false; flexcanConfig->enableDoze = false; flexcanConfig->enableMemoryErrorControl = true; flexcanConfig->enableNonCorrectableErrorEnterFreeze = true; flexcanConfig.timingConfig = timingConfig;

## Parameters

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

**22.2.6.8 void FLEXCAN\_SetTimingConfig ( CAN\_Type \* base, const flexcan\_timing\_config\_t \* pConfig )**

This function gives user settings to classical CAN or CAN FD nominal phase timing characteristic. The function is for an experienced user. For less experienced users, call the [FLEXCAN\\_SetBitRate\(\)](#) instead.

## Note

Calling [FLEXCAN\\_SetTimingConfig\(\)](#) overrides the bit rate set in [FLEXCAN\\_Init\(\)](#) or [FLEXCAN\\_SetBitRate\(\)](#).

## Parameters

|                |                                                |
|----------------|------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.               |
| <i>pConfig</i> | Pointer to the timing configuration structure. |

**22.2.6.9 status\_t FLEXCAN\_SetBitRate ( CAN\_Type \* base, uint32\_t sourceClock\_Hz, uint32\_t bitRate\_Bps )**

This function set the bit rate of classical CAN frame or CAN FD frame nominal phase base on [FLEXCAN\\_CalculateImprovedTimingValues\(\)](#) API calculated timing values.

## Note

Calling [FLEXCAN\\_SetBitRate\(\)](#) overrides the bit rate set in [FLEXCAN\\_Init\(\)](#).

## Parameters

|                        |                                  |
|------------------------|----------------------------------|
| <i>base</i>            | FlexCAN peripheral base address. |
| <i>sourceClock_-Hz</i> | Source Clock in Hz.              |
| <i>bitRate_Bps</i>     | Bit rate in Bps.                 |

## Returns

kStatus\_Success - Set CAN baud rate (only Nominal phase) successfully.

#### 22.2.6.10 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. |

#### 22.2.6.11 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.       |

#### 22.2.6.12 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.        |

### 22.2.6.13 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> |

### 22.2.6.14 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> |

### 22.2.6.15 void FLEXCAN\_SetRxFifoConfig ( CAN\_Type \* *base*, const flexcan\_rx\_fifo\_config\_t \* *pRxFifoConfig*, bool *enable* )

This function configures the FlexCAN Rx FIFO with given configuration.

## Note

Legacy Rx FIFO only can receive classic CAN message.

## Parameters

|                      |                                                                                                                                                           |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | FlexCAN peripheral base address.                                                                                                                          |
| <i>pRxFifoConfig</i> | Pointer to the FlexCAN Legacy Rx FIFO configuration structure. Can be NULL when enable parameter is false.                                                |
| <i>enable</i>        | Enable/disable Legacy Rx FIFO. <ul style="list-style-type: none"> <li>• true: Enable Legacy Rx FIFO.</li> <li>• false: Disable Legacy Rx FIFO.</li> </ul> |

#### 22.2.6.16 `static uint64_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`.

#### 22.2.6.17 `static void FLEXCAN_ClearStatusFlags ( CAN_Type * base, uint64_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 <code>_flexcan_flags</code> . |

**22.2.6.18** `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. |

**22.2.6.19** `static uint64_t FLEXCAN_GetMbStatusFlags ( CAN_Type * base, uint64_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.

**22.2.6.20** `static void FLEXCAN_ClearMbStatusFlags ( CAN_Type * base, uint64_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. |

**22.2.6.21 void FLEXCAN\_GetMemoryErrorReportStatus ( CAN\_Type \* *base*,  
flexcan\_memory\_error\_report\_status\_t \* *errorStatus* )**

This function gets the FlexCAN Memory Error Report registers status.

## Parameters

|                    |                                                                    |
|--------------------|--------------------------------------------------------------------|
| <i>base</i>        | FlexCAN peripheral base address.                                   |
| <i>errorStatus</i> | Pointer to FlexCAN Memory Error Report registers status structure. |

### 22.2.6.22 `static void FLEXCAN_EnableInterrupts ( CAN_Type * base, uint64_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](#).

## Parameters

|             |                                                                                     |
|-------------|-------------------------------------------------------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.                                                    |
| <i>mask</i> | The interrupts to enable. Logical OR of <a href="#">_flexcan_interrupt_enable</a> . |

### 22.2.6.23 `static void FLEXCAN_DisableInterrupts ( CAN_Type * base, uint64_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> . |

### 22.2.6.24 `static void FLEXCAN_EnableMblInterrupts ( CAN_Type * base, uint64_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. |

**22.2.6.25** `static void FLEXCAN_DisableMbInterrupts ( CAN_Type * base, uint64_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. |

#### 22.2.6.26 void FLEXCAN\_EnableRxFifoDMA ( CAN\_Type \* *base*, bool *enable* )

This function enables or disables the DMA feature of FlexCAN build-in Rx FIFO.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.  |
| <i>enable</i> | true to enable, false to disable. |

#### 22.2.6.27 static uintptr\_t FLEXCAN\_GetRxFifoHeadAddr ( CAN\_Type \* *base* ) [inline], [static]

This function returns the FlexCAN Rx FIFO Head address, which is mainly used for the DMA/eDMA use case.

Parameters

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

Returns

FlexCAN Rx FIFO Head address.

#### 22.2.6.28 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. |
|---------------|-----------------------------------|

### 22.2.6.29 **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. |

### 22.2.6.30 **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

|                        |                                                             |
|------------------------|-------------------------------------------------------------|
| <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.                                             |

### 22.2.6.31 `status_t FLEXCAN_ReadRxFifo ( CAN_Type * base, flexcan_frame_t * pRxFrame )`

This function reads a CAN message from the FlexCAN Legacy 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.                 |

### 22.2.6.32 `status_t FLEXCAN_TransferSendBlocking ( CAN_Type * base, uint8_t mbIdx, flexcan_frame_t * pTxFrame )`

Note

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. |
|---------------------|------------------------------------------|

### 22.2.6.33 `status_t FLEXCAN_TransferReceiveBlocking ( CAN_Type * base, uint8_t mbIdx, flexcan_frame_t * pRxFrame )`

Note

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>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.                                             |

### 22.2.6.34 `status_t FLEXCAN_TransferReceiveFifoBlocking ( CAN_Type * base, flexcan_frame_t * pRxFrame )`

Note

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.                 |

### 22.2.6.35 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. |

### 22.2.6.36 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.                          |

**22.2.6.37** `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> . |

## 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.                            |

### 22.2.6.38 `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.                  |

### 22.2.6.39 `status_t FLEXCAN_TransferGetReceiveFifoCount ( CAN_Type * base, flexcan_handle_t * handle, size_t * count )`

## Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                       |
| <i>handle</i> | FlexCAN handle pointer.                                                |
| <i>count</i>  | Number of CAN messages receive so far by the non-blocking transaction. |

Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_InvalidArgument</i> | count is Invalid.              |
| <i>kStatus_Success</i>         | Successfully return the count. |

#### 22.2.6.40 `uint32_t FLEXCAN_GetTimeStamp ( flexcan_handle_t * handle, uint8_t mblIdx )`

This 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>mblIdx</i> | The FlexCAN Message Buffer index. |

Return values

|            |                                                     |
|------------|-----------------------------------------------------|
| <i>the</i> | index of mailbox 's timestamp stored in the handle. |
|------------|-----------------------------------------------------|

#### 22.2.6.41 `void FLEXCAN_TransferAbortSend ( CAN_Type * base, flexcan_handle_t * handle, uint8_t mblIdx )`

This function aborts the interrupt driven message send process.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.  |
| <i>handle</i> | FlexCAN handle pointer.           |
| <i>mblIdx</i> | The FlexCAN Message Buffer index. |

**22.2.6.42** void FLEXCAN\_TransferAbortReceive ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, uint8\_t *mblidx* )

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. |

#### 22.2.6.43 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.          |

#### 22.2.6.44 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 23

# FlexIO: FlexIO Driver

### 23.1 Overview

The MCUXpresso SDK provides a generic driver and multiple protocol-specific FlexIO drivers for the FlexIO module of MCUXpresso SDK devices.

#### Modules

- [FlexIO Camera Driver](#)
- [FlexIO Driver](#)
- [FlexIO I2C Master Driver](#)
- [FlexIO I2S Driver](#)
- [FlexIO MCU Interface LCD Driver](#)
- [FlexIO SPI Driver](#)
- [FlexIO UART Driver](#)

## 23.2 FlexIO Driver

### 23.2.1 Overview

#### Data Structures

- struct `flexio_config_t`  
*Define FlexIO user configuration structure. [More...](#)*
- struct `flexio_timer_config_t`  
*Define FlexIO timer configuration structure. [More...](#)*
- struct `flexio_shifter_config_t`  
*Define FlexIO shifter configuration structure. [More...](#)*

#### Macros

- #define `FLEXIO_TIMER_TRIGGER_SEL_PININPUT(x)` `((uint32_t)(x) << 1U)`  
*Calculate FlexIO timer trigger.*

#### Typedefs

- typedef void(\* `flexio_isr_t`)(void \*base, void \*handle)  
*typedef for FlexIO simulated driver interrupt handler.*

#### Enumerations

- enum `flexio_timer_trigger_polarity_t` {  
  `kFLEXIO_TimerTriggerPolarityActiveHigh` = 0x0U,  
  `kFLEXIO_TimerTriggerPolarityActiveLow` = 0x1U }  
*Define time of timer trigger polarity.*
- enum `flexio_timer_trigger_source_t` {  
  `kFLEXIO_TimerTriggerSourceExternal` = 0x0U,  
  `kFLEXIO_TimerTriggerSourceInternal` = 0x1U }  
*Define type of timer trigger source.*
- enum `flexio_pin_config_t` {  
  `kFLEXIO_PinConfigOutputDisabled` = 0x0U,  
  `kFLEXIO_PinConfigOpenDrainOrBidirection` = 0x1U,  
  `kFLEXIO_PinConfigBidirectionOutputData` = 0x2U,  
  `kFLEXIO_PinConfigOutput` = 0x3U }  
*Define type of timer/shifter pin configuration.*
- enum `flexio_pin_polarity_t` {  
  `kFLEXIO_PinActiveHigh` = 0x0U,  
  `kFLEXIO_PinActiveLow` = 0x1U }  
*Definition of pin polarity.*

- enum flexio\_timer\_mode\_t {
  - kFLEXIO\_TimerModeDisabled = 0x0U,
  - kFLEXIO\_TimerModeDual8BitBaudBit = 0x1U,
  - kFLEXIO\_TimerModeDual8BitPWM = 0x2U,
  - kFLEXIO\_TimerModeSingle16Bit = 0x3U }

*Define type of timer work mode.*
- enum flexio\_timer\_output\_t {
  - kFLEXIO\_TimerOutputOneNotAffectedByReset = 0x0U,
  - kFLEXIO\_TimerOutputZeroNotAffectedByReset = 0x1U,
  - kFLEXIO\_TimerOutputOneAffectedByReset = 0x2U,
  - kFLEXIO\_TimerOutputZeroAffectedByReset = 0x3U }

*Define type of timer initial output or timer reset condition.*
- enum flexio\_timer\_decrement\_source\_t {
  - kFLEXIO\_TimerDecSrcOnFlexIOClockShiftTimerOutput = 0x0U,
  - kFLEXIO\_TimerDecSrcOnTriggerInputShiftTimerOutput,
  - kFLEXIO\_TimerDecSrcOnPinInputShiftPinInput,
  - kFLEXIO\_TimerDecSrcOnTriggerInputShiftTriggerInput }

*Define type of timer decrement.*
- enum flexio\_timer\_reset\_condition\_t {
  - kFLEXIO\_TimerResetNever = 0x0U,
  - kFLEXIO\_TimerResetOnTimerPinEqualToTimerOutput = 0x2U,
  - kFLEXIO\_TimerResetOnTimerTriggerEqualToTimerOutput = 0x3U,
  - kFLEXIO\_TimerResetOnTimerPinRisingEdge = 0x4U,
  - kFLEXIO\_TimerResetOnTimerTriggerRisingEdge = 0x6U,
  - kFLEXIO\_TimerResetOnTimerTriggerBothEdge = 0x7U }

*Define type of timer reset condition.*
- enum flexio\_timer\_disable\_condition\_t {
  - kFLEXIO\_TimerDisableNever = 0x0U,
  - kFLEXIO\_TimerDisableOnPreTimerDisable = 0x1U,
  - kFLEXIO\_TimerDisableOnTimerCompare = 0x2U,
  - kFLEXIO\_TimerDisableOnTimerCompareTriggerLow = 0x3U,
  - kFLEXIO\_TimerDisableOnPinBothEdge = 0x4U,
  - kFLEXIO\_TimerDisableOnPinBothEdgeTriggerHigh = 0x5U,
  - kFLEXIO\_TimerDisableOnTriggerFallingEdge = 0x6U }

*Define type of timer disable condition.*
- enum flexio\_timer\_enable\_condition\_t {
  - kFLEXIO\_TimerEnabledAlways = 0x0U,
  - kFLEXIO\_TimerEnableOnPrevTimerEnable = 0x1U,
  - kFLEXIO\_TimerEnableOnTriggerHigh = 0x2U,
  - kFLEXIO\_TimerEnableOnTriggerHighPinHigh = 0x3U,
  - kFLEXIO\_TimerEnableOnPinRisingEdge = 0x4U,
  - kFLEXIO\_TimerEnableOnPinRisingEdgeTriggerHigh = 0x5U,
  - kFLEXIO\_TimerEnableOnTriggerRisingEdge = 0x6U,
  - kFLEXIO\_TimerEnableOnTriggerBothEdge = 0x7U }

*Define type of timer enable condition.*
- enum flexio\_timer\_stop\_bit\_condition\_t {

```

kFLEXIO_TimerStopBitDisabled = 0x0U,
kFLEXIO_TimerStopBitEnableOnTimerCompare = 0x1U,
kFLEXIO_TimerStopBitEnableOnTimerDisable = 0x2U,
kFLEXIO_TimerStopBitEnableOnTimerCompareDisable = 0x3U }

```

*Define type of timer stop bit generate condition.*

- enum flexio\_timer\_start\_bit\_condition\_t {  
kFLEXIO\_TimerStartBitDisabled = 0x0U,  
kFLEXIO\_TimerStartBitEnabled = 0x1U }

*Define type of timer start bit generate condition.*

- enum flexio\_timer\_output\_state\_t {  
kFLEXIO\_PwmLow = 0,  
kFLEXIO\_PwmHigh }

*FlexIO as PWM channel output state.*

- enum flexio\_shifter\_timer\_polarity\_t {  
kFLEXIO\_ShifterTimerPolarityOnPositive = 0x0U,  
kFLEXIO\_ShifterTimerPolarityOnNegative = 0x1U }

*Define type of timer polarity for shifter control.*

- enum flexio\_shifter\_mode\_t {  
kFLEXIO\_ShifterDisabled = 0x0U,  
kFLEXIO\_ShifterModeReceive = 0x1U,  
kFLEXIO\_ShifterModeTransmit = 0x2U,  
kFLEXIO\_ShifterModeMatchStore = 0x4U,  
kFLEXIO\_ShifterModeMatchContinuous = 0x5U }

*Define type of shifter working mode.*

- enum flexio\_shifter\_input\_source\_t {  
kFLEXIO\_ShifterInputFromPin = 0x0U,  
kFLEXIO\_ShifterInputFromNextShifterOutput = 0x1U }

*Define type of shifter input source.*

- enum flexio\_shifter\_stop\_bit\_t {  
kFLEXIO\_ShifterStopBitDisable = 0x0U,  
kFLEXIO\_ShifterStopBitLow = 0x2U,  
kFLEXIO\_ShifterStopBitHigh = 0x3U }

*Define of STOP bit configuration.*

- enum flexio\_shifter\_start\_bit\_t {  
kFLEXIO\_ShifterStartBitDisabledLoadDataOnEnable = 0x0U,  
kFLEXIO\_ShifterStartBitDisabledLoadDataOnShift = 0x1U,  
kFLEXIO\_ShifterStartBitLow = 0x2U,  
kFLEXIO\_ShifterStartBitHigh = 0x3U }

*Define type of START bit configuration.*

- enum flexio\_shifter\_buffer\_type\_t {  
kFLEXIO\_ShifterBuffer = 0x0U,  
kFLEXIO\_ShifterBufferBitSwapped = 0x1U,  
kFLEXIO\_ShifterBufferByteSwapped = 0x2U,  
kFLEXIO\_ShifterBufferBitByteSwapped = 0x3U }

*Define FlexIO shifter buffer type.*

## Variables

- FLEXIO\_Type \*const `s_flexioBases` []  
*Pointers to flexio bases for each instance.*
- const `clock_ip_name_t s_flexioClocks` []  
*Pointers to flexio clocks for each instance.*

## Driver version

- #define `FSL_FLEXIO_DRIVER_VERSION` (MAKE\_VERSION(2, 2, 0))  
*FlexIO driver version.*

## FlexIO Initialization and De-initialization

- void `FLEXIO_GetDefaultConfig` (`flexio_config_t *userConfig`)  
*Gets the default configuration to configure the FlexIO module.*
- void `FLEXIO_Init` (FLEXIO\_Type \*base, const `flexio_config_t *userConfig`)  
*Configures the FlexIO with a FlexIO configuration.*
- void `FLEXIO_Deinit` (FLEXIO\_Type \*base)  
*Gates the FlexIO clock.*
- uint32\_t `FLEXIO_GetInstance` (FLEXIO\_Type \*base)  
*Get instance number for FLEXIO module.*

## FlexIO Basic Operation

- void `FLEXIO_Reset` (FLEXIO\_Type \*base)  
*Resets the FlexIO module.*
- static void `FLEXIO_Enable` (FLEXIO\_Type \*base, bool enable)  
*Enables the FlexIO module operation.*
- void `FLEXIO_SetShifterConfig` (FLEXIO\_Type \*base, uint8\_t index, const `flexio_shifter_config_t *shifterConfig`)  
*Configures the shifter with the shifter configuration.*
- void `FLEXIO_SetTimerConfig` (FLEXIO\_Type \*base, uint8\_t index, const `flexio_timer_config_t *timerConfig`)  
*Configures the timer with the timer configuration.*
- static void `FLEXIO_SetClockMode` (FLEXIO\_Type \*base, uint8\_t index, `flexio_timer_decrement_source_t` clocksource)  
*This function set the value of the prescaler on flexio channels.*

## FlexIO Interrupt Operation

- static void `FLEXIO_EnableShifterStatusInterrupts` (FLEXIO\_Type \*base, uint32\_t mask)  
*Enables the shifter status interrupt.*
- static void `FLEXIO_DisableShifterStatusInterrupts` (FLEXIO\_Type \*base, uint32\_t mask)  
*Disables the shifter status interrupt.*

- static void [FLEXIO\\_EnableShifterErrorInterrupts](#) (FLEXIO\_Type \*base, uint32\_t mask)  
*Enables the shifter error interrupt.*
- static void [FLEXIO\\_DisableShifterErrorInterrupts](#) (FLEXIO\_Type \*base, uint32\_t mask)  
*Disables the shifter error interrupt.*
- static void [FLEXIO\\_EnableTimerStatusInterrupts](#) (FLEXIO\_Type \*base, uint32\_t mask)  
*Enables the timer status interrupt.*
- static void [FLEXIO\\_DisableTimerStatusInterrupts](#) (FLEXIO\_Type \*base, uint32\_t mask)  
*Disables the timer status interrupt.*

## FlexIO Status Operation

- static uint32\_t [FLEXIO\\_GetShifterStatusFlags](#) (FLEXIO\_Type \*base)  
*Gets the shifter status flags.*
- static void [FLEXIO\\_ClearShifterStatusFlags](#) (FLEXIO\_Type \*base, uint32\_t mask)  
*Clears the shifter status flags.*
- static uint32\_t [FLEXIO\\_GetShifterErrorFlags](#) (FLEXIO\_Type \*base)  
*Gets the shifter error flags.*
- static void [FLEXIO\\_ClearShifterErrorFlags](#) (FLEXIO\_Type \*base, uint32\_t mask)  
*Clears the shifter error flags.*
- static uint32\_t [FLEXIO\\_GetTimerStatusFlags](#) (FLEXIO\_Type \*base)  
*Gets the timer status flags.*
- static void [FLEXIO\\_ClearTimerStatusFlags](#) (FLEXIO\_Type \*base, uint32\_t mask)  
*Clears the timer status flags.*

## FlexIO DMA Operation

- static void [FLEXIO\\_EnableShifterStatusDMA](#) (FLEXIO\_Type \*base, uint32\_t mask, bool enable)  
*Enables/disables the shifter status DMA.*
- uint32\_t [FLEXIO\\_GetShifterBufferAddress](#) (FLEXIO\_Type \*base, [flexio\\_shifter\\_buffer\\_type\\_t](#) type, uint8\_t index)  
*Gets the shifter buffer address for the DMA transfer usage.*
- [status\\_t](#) [FLEXIO\\_RegisterHandleIRQ](#) (void \*base, void \*handle, [flexio\\_isr\\_t](#) isr)  
*Registers the handle and the interrupt handler for the FlexIO-simulated peripheral.*
- [status\\_t](#) [FLEXIO\\_UnregisterHandleIRQ](#) (void \*base)  
*Unregisters the handle and the interrupt handler for the FlexIO-simulated peripheral.*

## 23.2.2 Data Structure Documentation

### 23.2.2.1 struct flexio\_config\_t

#### Data Fields

- bool [enableFlexio](#)  
*Enable/disable FlexIO module.*
- bool [enableInDoze](#)  
*Enable/disable FlexIO operation in doze mode.*

- bool [enableInDebug](#)  
*Enable/disable FlexIO operation in debug mode.*
- bool [enableFastAccess](#)  
*Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*

## Field Documentation

### (1) bool flexio\_config\_t::enableFastAccess

#### 23.2.2.2 struct flexio\_timer\_config\_t

### Data Fields

- uint32\_t [triggerSelect](#)  
*The internal trigger selection number using MACROs.*
- [flexio\\_timer\\_trigger\\_polarity\\_t](#) triggerPolarity  
*Trigger Polarity.*
- [flexio\\_timer\\_trigger\\_source\\_t](#) triggerSource  
*Trigger Source, internal (see 'trgsel') or external.*
- [flexio\\_pin\\_config\\_t](#) pinConfig  
*Timer Pin Configuration.*
- uint32\_t [pinSelect](#)  
*Timer Pin number Select.*
- [flexio\\_pin\\_polarity\\_t](#) pinPolarity  
*Timer Pin Polarity.*
- [flexio\\_timer\\_mode\\_t](#) timerMode  
*Timer work Mode.*
- [flexio\\_timer\\_output\\_t](#) timerOutput  
*Configures the initial state of the Timer Output and whether it is affected by the Timer reset.*
- [flexio\\_timer\\_decrement\\_source\\_t](#) timerDecrement  
*Configures the source of the Timer decrement and the source of the Shift clock.*
- [flexio\\_timer\\_reset\\_condition\\_t](#) timerReset  
*Configures the condition that causes the timer counter (and optionally the timer output) to be reset.*
- [flexio\\_timer\\_disable\\_condition\\_t](#) timerDisable  
*Configures the condition that causes the Timer to be disabled and stop decrementing.*
- [flexio\\_timer\\_enable\\_condition\\_t](#) timerEnable  
*Configures the condition that causes the Timer to be enabled and start decrementing.*
- [flexio\\_timer\\_stop\\_bit\\_condition\\_t](#) timerStop  
*Timer STOP Bit generation.*
- [flexio\\_timer\\_start\\_bit\\_condition\\_t](#) timerStart  
*Timer STRAT Bit generation.*
- uint32\_t [timerCompare](#)  
*Value for Timer Compare N Register.*

## Field Documentation

- (1) `uint32_t flexio_timer_config_t::triggerSelect`
- (2) `flexio_timer_trigger_polarity_t flexio_timer_config_t::triggerPolarity`
- (3) `flexio_timer_trigger_source_t flexio_timer_config_t::triggerSource`
- (4) `flexio_pin_config_t flexio_timer_config_t::pinConfig`
- (5) `uint32_t flexio_timer_config_t::pinSelect`
- (6) `flexio_pin_polarity_t flexio_timer_config_t::pinPolarity`
- (7) `flexio_timer_mode_t flexio_timer_config_t::timerMode`
- (8) `flexio_timer_output_t flexio_timer_config_t::timerOutput`
- (9) `flexio_timer_decrement_source_t flexio_timer_config_t::timerDecrement`
- (10) `flexio_timer_reset_condition_t flexio_timer_config_t::timerReset`
- (11) `flexio_timer_disable_condition_t flexio_timer_config_t::timerDisable`
- (12) `flexio_timer_enable_condition_t flexio_timer_config_t::timerEnable`
- (13) `flexio_timer_stop_bit_condition_t flexio_timer_config_t::timerStop`
- (14) `flexio_timer_start_bit_condition_t flexio_timer_config_t::timerStart`
- (15) `uint32_t flexio_timer_config_t::timerCompare`

### 23.2.2.3 struct flexio\_shifter\_config\_t

#### Data Fields

- `uint32_t timerSelect`  
Selects which Timer is used for controlling the logic/shift register and generating the Shift clock.
- `flexio_shifter_timer_polarity_t timerPolarity`  
Timer Polarity.
- `flexio_pin_config_t pinConfig`  
Shifter Pin Configuration.
- `uint32_t pinSelect`  
Shifter Pin number Select.
- `flexio_pin_polarity_t pinPolarity`  
Shifter Pin Polarity.
- `flexio_shifter_mode_t shifterMode`  
Configures the mode of the Shifter.
- `flexio_shifter_input_source_t inputSource`  
Selects the input source for the shifter.



- `flexio_shifter_stop_bit_t` `shifterStop`  
*Shifter STOP bit.*
- `flexio_shifter_start_bit_t` `shifterStart`  
*Shifter START bit.*

### Field Documentation

- (1) `uint32_t flexio_shifter_config_t::timerSelect`
- (2) `flexio_shifter_timer_polarity_t flexio_shifter_config_t::timerPolarity`
- (3) `flexio_pin_config_t flexio_shifter_config_t::pinConfig`
- (4) `uint32_t flexio_shifter_config_t::pinSelect`
- (5) `flexio_pin_polarity_t flexio_shifter_config_t::pinPolarity`
- (6) `flexio_shifter_mode_t flexio_shifter_config_t::shifterMode`
- (7) `flexio_shifter_input_source_t flexio_shifter_config_t::inputSource`
- (8) `flexio_shifter_stop_bit_t flexio_shifter_config_t::shifterStop`
- (9) `flexio_shifter_start_bit_t flexio_shifter_config_t::shifterStart`

### 23.2.3 Macro Definition Documentation

23.2.3.1 `#define FSL_FLEXIO_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))`

23.2.3.2 `#define FLEXIO_TIMER_TRIGGER_SEL_PININPUT( x ) ((uint32_t)(x) << 1U)`

### 23.2.4 Typedef Documentation

23.2.4.1 `typedef void(* flexio_isr_t)(void *base, void *handle)`

### 23.2.5 Enumeration Type Documentation

23.2.5.1 `enum flexio_timer_trigger_polarity_t`

Enumerator

*kFLEXIO\_TimerTriggerPolarityActiveHigh* Active high.

*kFLEXIO\_TimerTriggerPolarityActiveLow* Active low.

### 23.2.5.2 enum flexio\_timer\_trigger\_source\_t

Enumerator

*kFLEXIO\_TimerTriggerSourceExternal* External trigger selected.

*kFLEXIO\_TimerTriggerSourceInternal* Internal trigger selected.

### 23.2.5.3 enum flexio\_pin\_config\_t

Enumerator

*kFLEXIO\_PinConfigOutputDisabled* Pin output disabled.

*kFLEXIO\_PinConfigOpenDrainOrBidirection* Pin open drain or bidirectional output enable.

*kFLEXIO\_PinConfigBidirectionOutputData* Pin bidirectional output data.

*kFLEXIO\_PinConfigOutput* Pin output.

### 23.2.5.4 enum flexio\_pin\_polarity\_t

Enumerator

*kFLEXIO\_PinActiveHigh* Active high.

*kFLEXIO\_PinActiveLow* Active low.

### 23.2.5.5 enum flexio\_timer\_mode\_t

Enumerator

*kFLEXIO\_TimerModeDisabled* Timer Disabled.

*kFLEXIO\_TimerModeDual8BitBaudBit* Dual 8-bit counters baud/bit mode.

*kFLEXIO\_TimerModeDual8BitPWM* Dual 8-bit counters PWM mode.

*kFLEXIO\_TimerModeSingle16Bit* Single 16-bit counter mode.

### 23.2.5.6 enum flexio\_timer\_output\_t

Enumerator

*kFLEXIO\_TimerOutputOneNotAffectedByReset* Logic one when enabled and is not affected by timer reset.

*kFLEXIO\_TimerOutputZeroNotAffectedByReset* Logic zero when enabled and is not affected by timer reset.

*kFLEXIO\_TimerOutputOneAffectedByReset* Logic one when enabled and on timer reset.

*kFLEXIO\_TimerOutputZeroAffectedByReset* Logic zero when enabled and on timer reset.

### 23.2.5.7 enum flexio\_timer\_decrement\_source\_t

Enumerator

- kFLEXIO\_TimerDecSrcOnFlexIOClockShiftTimerOutput* Decrement counter on FlexIO clock, Shift clock equals Timer output.
- kFLEXIO\_TimerDecSrcOnTriggerInputShiftTimerOutput* Decrement counter on Trigger input (both edges), Shift clock equals Timer output.
- kFLEXIO\_TimerDecSrcOnPinInputShiftPinInput* Decrement counter on Pin input (both edges), Shift clock equals Pin input.
- kFLEXIO\_TimerDecSrcOnTriggerInputShiftTriggerInput* Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

### 23.2.5.8 enum flexio\_timer\_reset\_condition\_t

Enumerator

- kFLEXIO\_TimerResetNever* Timer never reset.
- kFLEXIO\_TimerResetOnTimerPinEqualToTimerOutput* Timer reset on Timer Pin equal to Timer Output.
- kFLEXIO\_TimerResetOnTimerTriggerEqualToTimerOutput* Timer reset on Timer Trigger equal to Timer Output.
- kFLEXIO\_TimerResetOnTimerPinRisingEdge* Timer reset on Timer Pin rising edge.
- kFLEXIO\_TimerResetOnTimerTriggerRisingEdge* Timer reset on Trigger rising edge.
- kFLEXIO\_TimerResetOnTimerTriggerBothEdge* Timer reset on Trigger rising or falling edge.

### 23.2.5.9 enum flexio\_timer\_disable\_condition\_t

Enumerator

- kFLEXIO\_TimerDisableNever* Timer never disabled.
- kFLEXIO\_TimerDisableOnPreTimerDisable* Timer disabled on Timer N-1 disable.
- kFLEXIO\_TimerDisableOnTimerCompare* Timer disabled on Timer compare.
- kFLEXIO\_TimerDisableOnTimerCompareTriggerLow* Timer disabled on Timer compare and Trigger Low.
- kFLEXIO\_TimerDisableOnPinBothEdge* Timer disabled on Pin rising or falling edge.
- kFLEXIO\_TimerDisableOnPinBothEdgeTriggerHigh* Timer disabled on Pin rising or falling edge provided Trigger is high.
- kFLEXIO\_TimerDisableOnTriggerFallingEdge* Timer disabled on Trigger falling edge.

### 23.2.5.10 enum flexio\_timer\_enable\_condition\_t

Enumerator

- kFLEXIO\_TimerEnabledAlways* Timer always enabled.

*kFLEXIO\_TimerEnableOnPrevTimerEnable* Timer enabled on Timer N-1 enable.  
*kFLEXIO\_TimerEnableOnTriggerHigh* Timer enabled on Trigger high.  
*kFLEXIO\_TimerEnableOnTriggerHighPinHigh* Timer enabled on Trigger high and Pin high.  
*kFLEXIO\_TimerEnableOnPinRisingEdge* Timer enabled on Pin rising edge.  
*kFLEXIO\_TimerEnableOnPinRisingEdgeTriggerHigh* Timer enabled on Pin rising edge and Trigger high.  
*kFLEXIO\_TimerEnableOnTriggerRisingEdge* Timer enabled on Trigger rising edge.  
*kFLEXIO\_TimerEnableOnTriggerBothEdge* Timer enabled on Trigger rising or falling edge.

### 23.2.5.11 enum flexio\_timer\_stop\_bit\_condition\_t

Enumerator

*kFLEXIO\_TimerStopBitDisabled* Stop bit disabled.  
*kFLEXIO\_TimerStopBitEnableOnTimerCompare* Stop bit is enabled on timer compare.  
*kFLEXIO\_TimerStopBitEnableOnTimerDisable* Stop bit is enabled on timer disable.  
*kFLEXIO\_TimerStopBitEnableOnTimerCompareDisable* Stop bit is enabled on timer compare and timer disable.

### 23.2.5.12 enum flexio\_timer\_start\_bit\_condition\_t

Enumerator

*kFLEXIO\_TimerStartBitDisabled* Start bit disabled.  
*kFLEXIO\_TimerStartBitEnabled* Start bit enabled.

### 23.2.5.13 enum flexio\_timer\_output\_state\_t

Enumerator

*kFLEXIO\_PwmLow* The output state of PWM channel is low.  
*kFLEXIO\_PwmHigh* The output state of PWM channel is high.

### 23.2.5.14 enum flexio\_shifter\_timer\_polarity\_t

Enumerator

*kFLEXIO\_ShifterTimerPolarityOnPositive* Shift on positive edge of shift clock.  
*kFLEXIO\_ShifterTimerPolarityOnNegative* Shift on negative edge of shift clock.

**23.2.5.15 enum flexio\_shifter\_mode\_t**

Enumerator

*kFLEXIO\_ShifterDisabled* Shifter is disabled.  
*kFLEXIO\_ShifterModeReceive* Receive mode.  
*kFLEXIO\_ShifterModeTransmit* Transmit mode.  
*kFLEXIO\_ShifterModeMatchStore* Match store mode.  
*kFLEXIO\_ShifterModeMatchContinuous* Match continuous mode.

**23.2.5.16 enum flexio\_shifter\_input\_source\_t**

Enumerator

*kFLEXIO\_ShifterInputFromPin* Shifter input from pin.  
*kFLEXIO\_ShifterInputFromNextShifterOutput* Shifter input from Shifter N+1.

**23.2.5.17 enum flexio\_shifter\_stop\_bit\_t**

Enumerator

*kFLEXIO\_ShifterStopBitDisable* Disable shifter stop bit.  
*kFLEXIO\_ShifterStopBitLow* Set shifter stop bit to logic low level.  
*kFLEXIO\_ShifterStopBitHigh* Set shifter stop bit to logic high level.

**23.2.5.18 enum flexio\_shifter\_start\_bit\_t**

Enumerator

*kFLEXIO\_ShifterStartBitDisabledLoadDataOnEnable* Disable shifter start bit, transmitter loads data on enable.  
*kFLEXIO\_ShifterStartBitDisabledLoadDataOnShift* Disable shifter start bit, transmitter loads data on first shift.  
*kFLEXIO\_ShifterStartBitLow* Set shifter start bit to logic low level.  
*kFLEXIO\_ShifterStartBitHigh* Set shifter start bit to logic high level.

**23.2.5.19 enum flexio\_shifter\_buffer\_type\_t**

Enumerator

*kFLEXIO\_ShifterBuffer* Shifter Buffer N Register.  
*kFLEXIO\_ShifterBufferBitSwapped* Shifter Buffer N Bit Byte Swapped Register.  
*kFLEXIO\_ShifterBufferByteSwapped* Shifter Buffer N Byte Swapped Register.  
*kFLEXIO\_ShifterBufferBitByteSwapped* Shifter Buffer N Bit Swapped Register.

## 23.2.6 Function Documentation

### 23.2.6.1 void FLEXIO\_GetDefaultConfig ( flexio\_config\_t \* userConfig )

The configuration can be used directly to call the FLEXIO\_Configure().

Example:

```
flexio_config_t config;
FLEXIO_GetDefaultConfig(&config);
```

Parameters

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>userConfig</i> | pointer to <a href="#">flexio_config_t</a> structure |
|-------------------|------------------------------------------------------|

### 23.2.6.2 void FLEXIO\_Init ( FLEXIO\_Type \* base, const flexio\_config\_t \* userConfig )

The configuration structure can be filled by the user or be set with default values by [FLEXIO\\_GetDefaultConfig\(\)](#).

Example

```
flexio_config_t config = {
.enableFlexio = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false
};
FLEXIO_Configure(base, &config);
```

Parameters

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>base</i>       | FlexIO peripheral base address                       |
| <i>userConfig</i> | pointer to <a href="#">flexio_config_t</a> structure |

### 23.2.6.3 void FLEXIO\_Deinit ( FLEXIO\_Type \* base )

Call this API to stop the FlexIO clock.

Note

After calling this API, call the FLEXIO\_Init to use the FlexIO module.

## Parameters

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

**23.2.6.4 uint32\_t FLEXIO\_GetInstance ( FLEXIO\_Type \* *base* )**

## Parameters

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

**23.2.6.5 void FLEXIO\_Reset ( FLEXIO\_Type \* *base* )**

## Parameters

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

**23.2.6.6 static void FLEXIO\_Enable ( FLEXIO\_Type \* *base*, bool *enable* ) [inline], [static]**

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FlexIO peripheral base address    |
| <i>enable</i> | true to enable, false to disable. |

**23.2.6.7 void FLEXIO\_SetShifterConfig ( FLEXIO\_Type \* *base*, uint8\_t *index*, const flexio\_shifter\_config\_t \* *shifterConfig* )**

The configuration structure covers both the SHIFTCTL and SHIFTCFG registers. To configure the shifter to the proper mode, select which timer controls the shifter to shift, whether to generate start bit/stop bit, and the polarity of start bit and stop bit.

## Example

```
flexio_shifter_config_t config = {
 .timerSelect = 0,
 .timerPolarity = kFLEXIO_ShifterTimerPolarityOnPositive,
 .pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
 .pinPolarity = kFLEXIO_PinActiveLow,
 .shifterMode = kFLEXIO_ShifterModeTransmit,
 .inputSource = kFLEXIO_ShifterInputFromPin,
 .shifterStop = kFLEXIO_ShifterStopBitHigh,
 .shifterStart = kFLEXIO_ShifterStartBitLow
};
FLEXIO_SetShifterConfig(base, &config);
```

## Parameters

|                      |                                                              |
|----------------------|--------------------------------------------------------------|
| <i>base</i>          | FlexIO peripheral base address                               |
| <i>index</i>         | Shifter index                                                |
| <i>shifterConfig</i> | Pointer to <a href="#">flexio_shifter_config_t</a> structure |

### 23.2.6.8 void FLEXIO\_SetTimerConfig ( FLEXIO\_Type \* *base*, uint8\_t *index*, const flexio\_timer\_config\_t \* *timerConfig* )

The configuration structure covers both the TIMCTL and TIMCFG registers. To configure the timer to the proper mode, select trigger source for timer and the timer pin output and the timing for timer.

## Example

```
flexio_timer_config_t config = {
 .triggerSelect = FLEXIO_TIMER_TRIGGER_SEL_SHIFTnSTAT(0),
 .triggerPolarity = kFLEXIO_TimerTriggerPolarityActiveLow,
 .triggerSource = kFLEXIO_TimerTriggerSourceInternal,
 .pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
 .pinSelect = 0,
 .pinPolarity = kFLEXIO_PinActiveHigh,
 .timerMode = kFLEXIO_TimerModeDual8BitBaudBit,
 .timerOutput = kFLEXIO_TimerOutputZeroNotAffectedByReset,
 .timerDecrement = kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput
 ,
 .timerReset = kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput,
 .timerDisable = kFLEXIO_TimerDisableOnTimerCompare,
 .timerEnable = kFLEXIO_TimerEnableOnTriggerHigh,
 .timerStop = kFLEXIO_TimerStopBitEnableOnTimerDisable,
 .timerStart = kFLEXIO_TimerStartBitEnabled
};
FLEXIO_SetTimerConfig(base, &config);
```

## Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>base</i>        | FlexIO peripheral base address                                 |
| <i>index</i>       | Timer index                                                    |
| <i>timerConfig</i> | Pointer to the <a href="#">flexio_timer_config_t</a> structure |

### 23.2.6.9 static void FLEXIO\_SetClockMode ( FLEXIO\_Type \* *base*, uint8\_t *index*, flexio\_timer\_decrement\_source\_t *clocksource* ) [inline], [static]



Parameters

|                    |                                                  |
|--------------------|--------------------------------------------------|
| <i>base</i>        | Pointer to the FlexIO simulated peripheral type. |
| <i>clocksource</i> | Set clock value                                  |

**23.2.6.10 static void FLEXIO\_EnableShifterStatusInterrupts ( FLEXIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

The interrupt generates when the corresponding SSF is set.

Parameters

|             |                                                                                   |
|-------------|-----------------------------------------------------------------------------------|
| <i>base</i> | FlexIO peripheral base address                                                    |
| <i>mask</i> | The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$ |

Note

For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

**23.2.6.11 static void FLEXIO\_DisableShifterStatusInterrupts ( FLEXIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

The interrupt won't generate when the corresponding SSF is set.

Parameters

|             |                                                                                   |
|-------------|-----------------------------------------------------------------------------------|
| <i>base</i> | FlexIO peripheral base address                                                    |
| <i>mask</i> | The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$ |

Note

For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

**23.2.6.12 static void FLEXIO\_EnableShifterErrorInterrupts ( FLEXIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

The interrupt generates when the corresponding SEF is set.

## Parameters

|             |                                                                                  |
|-------------|----------------------------------------------------------------------------------|
| <i>base</i> | FlexIO peripheral base address                                                   |
| <i>mask</i> | The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$ |

## Note

For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

### 23.2.6.13 static void FLEXIO\_DisableShifterErrorInterrupts ( FLEXIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

The interrupt won't generate when the corresponding SEF is set.

## Parameters

|             |                                                                                  |
|-------------|----------------------------------------------------------------------------------|
| <i>base</i> | FlexIO peripheral base address                                                   |
| <i>mask</i> | The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$ |

## Note

For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

### 23.2.6.14 static void FLEXIO\_EnableTimerStatusInterrupts ( FLEXIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

The interrupt generates when the corresponding SSF is set.

## Parameters

|             |                                                                               |
|-------------|-------------------------------------------------------------------------------|
| <i>base</i> | FlexIO peripheral base address                                                |
| <i>mask</i> | The timer status mask which can be calculated by $(1 \ll \text{timer index})$ |

## Note

For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using  $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

**23.2.6.15** `static void FLEXIO_DisableTimerStatusInterrupts ( FLEXIO_Type * base,  
uint32_t mask ) [inline], [static]`

The interrupt won't generate when the corresponding SSF is set.

## Parameters

|             |                                                                               |
|-------------|-------------------------------------------------------------------------------|
| <i>base</i> | FlexIO peripheral base address                                                |
| <i>mask</i> | The timer status mask which can be calculated by $(1 \ll \text{timer index})$ |

## Note

For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using  $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

### 23.2.6.16 `static uint32_t FLEXIO_GetShifterStatusFlags ( FLEXIO_Type * base )` `[inline], [static]`

## Parameters

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

## Returns

Shifter status flags

### 23.2.6.17 `static void FLEXIO_ClearShifterStatusFlags ( FLEXIO_Type * base, uint32_t mask )` `[inline], [static]`

## Parameters

|             |                                                                                   |
|-------------|-----------------------------------------------------------------------------------|
| <i>base</i> | FlexIO peripheral base address                                                    |
| <i>mask</i> | The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$ |

## Note

For clearing multiple shifter status flags, for example, two shifter status flags, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

### 23.2.6.18 `static uint32_t FLEXIO_GetShifterErrorFlags ( FLEXIO_Type * base )` `[inline], [static]`

## Parameters

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

## Returns

Shifter error flags

### 23.2.6.19 static void FLEXIO\_ClearShifterErrorFlags ( FLEXIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                                  |
|-------------|----------------------------------------------------------------------------------|
| <i>base</i> | FlexIO peripheral base address                                                   |
| <i>mask</i> | The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$ |

## Note

For clearing multiple shifter error flags, for example, two shifter error flags, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

### 23.2.6.20 static uint32\_t FLEXIO\_GetTimerStatusFlags ( FLEXIO\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

Timer status flags

### 23.2.6.21 static void FLEXIO\_ClearTimerStatusFlags ( FLEXIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                               |
|-------------|-------------------------------------------------------------------------------|
| <i>base</i> | FlexIO peripheral base address                                                |
| <i>mask</i> | The timer status mask which can be calculated by $(1 \ll \text{timer index})$ |

## Note

For clearing multiple timer status flags, for example, two timer status flags, can calculate the mask by using  $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

### 23.2.6.22 static void FLEXIO\_EnableShifterStatusDMA ( FLEXIO\_Type \* *base*, uint32\_t *mask*, bool *enable* ) [inline], [static]

The DMA request generates when the corresponding SSF is set.

## Note

For multiple shifter status DMA enables, for example, calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

## Parameters

|               |                                                                                   |
|---------------|-----------------------------------------------------------------------------------|
| <i>base</i>   | FlexIO peripheral base address                                                    |
| <i>mask</i>   | The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$ |
| <i>enable</i> | True to enable, false to disable.                                                 |

### 23.2.6.23 uint32\_t FLEXIO\_GetShifterBufferAddress ( FLEXIO\_Type \* *base*, flexio\_shifter\_buffer\_type\_t *type*, uint8\_t *index* )

## Parameters

|              |                                              |
|--------------|----------------------------------------------|
| <i>base</i>  | FlexIO peripheral base address               |
| <i>type</i>  | Shifter type of flexio_shifter_buffer_type_t |
| <i>index</i> | Shifter index                                |

## Returns

Corresponding shifter buffer index

### 23.2.6.24 status\_t FLEXIO\_RegisterHandleIRQ ( void \* *base*, void \* *handle*, flexio\_isr\_t *isr* )

## Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | Pointer to the FlexIO simulated peripheral type.        |
| <i>handle</i> | Pointer to the handler for FlexIO simulated peripheral. |
| <i>isr</i>    | FlexIO simulated peripheral interrupt handler.          |

## Return values

|                           |                                                |
|---------------------------|------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                |
| <i>kStatus_OutOfRange</i> | The FlexIO type/handle/ISR table out of range. |

**23.2.6.25 status\_t FLEXIO\_UnregisterHandleIRQ ( void \* base )**

## Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| <i>base</i> | Pointer to the FlexIO simulated peripheral type. |
|-------------|--------------------------------------------------|

## Return values

|                           |                                                |
|---------------------------|------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                |
| <i>kStatus_OutOfRange</i> | The FlexIO type/handle/ISR table out of range. |

**23.2.7 Variable Documentation****23.2.7.1 FLEXIO\_Type\* const s\_flexioBases[]****23.2.7.2 const clock\_ip\_name\_t s\_flexioClocks[]**

## 23.3 FlexIO Camera Driver

### 23.3.1 Overview

The MCUXpresso SDK provides a driver for the camera function using Flexible I/O.

FlexIO Camera driver includes functional APIs and eDMA transactional APIs. Functional APIs target low level APIs. Users can use functional APIs for FlexIO Camera initialization/configuration/operation purpose. Using the functional API requires knowledge of the FlexIO Camera peripheral and how to organize functional APIs to meet the requirements of the application. All functional API use the [FLEXIO\\_CAMERA\\_Type](#) \* as the first parameter. FlexIO Camera functional operation groups provide the functional APIs set.

eDMA transactional APIs target high-level APIs. Users can use the transactional API to enable the peripheral quickly and can also use in the application if the code size and performance of transactional APIs satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the `flexio_camera_edma_handle_t` as the second parameter. Users need to initialize the handle by calling the [FLEXIO\\_CAMERA\\_TransferCreateHandleEDMA\(\)](#) API.

eDMA transactional APIs support asynchronous receive. This means that the functions [FLEXIO\\_CAMERA\\_TransferReceiveEDMA\(\)](#) set up an interrupt for data receive. When the receive is complete, the upper layer is notified through a callback function with the status `kStatus_FLEXIO_CAMERA_RxIdle`.

### 23.3.2 Typical use case

#### 23.3.2.1 FlexIO Camera Receive using eDMA method

```
volatile uint32_t isEDMAGetOnePictureFinish = false;
edma_handle_t g_edmaHandle;
flexio_camera_edma_handle_t g_cameraEdmaHandle;
edma_config_t edmaConfig;
FLEXIO_CAMERA_Type g_FlexioCameraDevice = {.flexioBase = FLEXIO0,
 .datPinStartIdx = 24U, /* fxio_pin 24 -31 are used. */
 .pclkPinIdx = 1U, /* fxio_pin 1 is used as pclk pin. */
 .hrefPinIdx = 18U, /* flexio_pin 18 is used as href pin. */
 .shifterStartIdx = 0U, /* Shifter 0 = 7 are used. */
 .shifterCount = 8U,
 .timerIdx = 0U};

flexio_camera_config_t cameraConfig;

/* Configure DMAMUX */
DMAMUX_Init(DMAMUX0);
/* Configure DMA */
EDMA_GetDefaultConfig(&edmaConfig);
EDMA_Init(DMA0, &edmaConfig);

DMAMUX_SetSource(DMAMUX0, DMA_CHN_FLEXIO_TO_FRAMEBUFF, (g_FlexioCameraDevice.
 shifterStartIdx + 1U));
DMAMUX_EnableChannel(DMAMUX0, DMA_CHN_FLEXIO_TO_FRAMEBUFF);
EDMA_CreateHandle(&g_edmaHandle, DMA0, DMA_CHN_FLEXIO_TO_FRAMEBUFF);

FLEXIO_CAMERA_GetDefaultConfig(&cameraConfig);
FLEXIO_CAMERA_Init(&g_FlexioCameraDevice, &cameraConfig);
/* Clear all the flag. */
```



```

FLEXIO_CAMERA_ClearStatusFlags(&g_FlexioCameraDevice,
 kFLEXIO_CAMERA_RxDataRegFullFlag |
 kFLEXIO_CAMERA_RxErrorFlag);
FLEXIO_ClearTimerStatusFlags(FLEXIO0, 0xFF);
FLEXIO_CAMERA_TransferCreateHandleEDMA(&g_FlexioCameraDevice, &
 g_cameraEdmaHandle, FLEXIO_CAMERA_UserCallback, NULL,
 &g_edmaHandle);
cameraTransfer.dataAddress = (uint32_t)u16CameraFrameBuffer;
cameraTransfer.dataNum = sizeof(u16CameraFrameBuffer);
FLEXIO_CAMERA_TransferReceiveEDMA(&g_FlexioCameraDevice, &
 g_cameraEdmaHandle, &cameraTransfer);
while (!(isEDMAGetOnePictureFinish))
{
 ;
}

/* A callback function is also needed */
void FLEXIO_CAMERA_UserCallback(FLEXIO_CAMERA_Type *base,
 flexio_camera_edma_handle_t *handle,
 status_t status,
 void *userData)
{
 userData = userData;
 /* eDMA Transfer finished */
 if (kStatus_FLEXIO_CAMERA_RxIdle == status)
 {
 isEDMAGetOnePictureFinish = true;
 }
}

```

## Modules

- [FlexIO eDMA Camera Driver](#)

## Data Structures

- struct [FLEXIO\\_CAMERA\\_Type](#)  
Define structure of configuring the FlexIO Camera device. [More...](#)
- struct [flexio\\_camera\\_config\\_t](#)  
Define FlexIO Camera user configuration structure. [More...](#)
- struct [flexio\\_camera\\_transfer\\_t](#)  
Define FlexIO Camera transfer structure. [More...](#)

## Macros

- #define [FLEXIO\\_CAMERA\\_PARALLEL\\_DATA\\_WIDTH](#) (8U)  
Define the Camera CPI interface is constantly 8-bit width.

## Enumerations

- enum {  
    [kStatus\\_FLEXIO\\_CAMERA\\_RxBusy](#) = MAKE\_STATUS(kStatusGroup\_FLEXIO\_CAMERA,

```

0),
kStatus_FLEXIO_CAMERA_RxIdle = MAKE_STATUS(kStatusGroup_FLEXIO_CAMERA, 1)
}

```

*Error codes for the Camera driver.*

- enum `_flexio_camera_status_flags` {
 `kFLEXIO_CAMERA_RxDataRegFullFlag = 0x1U,`  
`kFLEXIO_CAMERA_RxErrorFlag = 0x2U }`

*Define FlexIO Camera status mask.*

## Driver version

- #define `FSL_FLEXIO_CAMERA_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 3)`)  
*FlexIO Camera driver version 2.1.3.*

## Initialization and configuration

- void `FLEXIO_CAMERA_Init` (`FLEXIO_CAMERA_Type *base`, const `flexio_camera_config_t *config`)  
*Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO Camera.*
- void `FLEXIO_CAMERA_Deinit` (`FLEXIO_CAMERA_Type *base`)  
*Resets the FLEXIO\_CAMERA shifer and timer config.*
- void `FLEXIO_CAMERA_GetDefaultConfig` (`flexio_camera_config_t *config`)  
*Gets the default configuration to configure the FlexIO Camera.*
- static void `FLEXIO_CAMERA_Enable` (`FLEXIO_CAMERA_Type *base`, bool enable)  
*Enables/disables the FlexIO Camera module operation.*

## Status

- uint32\_t `FLEXIO_CAMERA_GetStatusFlags` (`FLEXIO_CAMERA_Type *base`)  
*Gets the FlexIO Camera status flags.*
- void `FLEXIO_CAMERA_ClearStatusFlags` (`FLEXIO_CAMERA_Type *base`, uint32\_t mask)  
*Clears the receive buffer full flag manually.*

## Interrupts

- void `FLEXIO_CAMERA_EnableInterrupt` (`FLEXIO_CAMERA_Type *base`)  
*Switches on the interrupt for receive buffer full event.*
- void `FLEXIO_CAMERA_DisableInterrupt` (`FLEXIO_CAMERA_Type *base`)  
*Switches off the interrupt for receive buffer full event.*

## DMA support

- static void `FLEXIO_CAMERA_EnableRxDMA` (`FLEXIO_CAMERA_Type *base`, bool enable)

*Enables/disables the FlexIO Camera receive DMA.*

- static uint32\_t `FLEXIO_CAMERA_GetRxBufferAddress` (`FLEXIO_CAMERA_Type *base`)  
*Gets the data from the receive buffer.*

### 23.3.3 Data Structure Documentation

#### 23.3.3.1 struct FLEXIO\_CAMERA\_Type

##### Data Fields

- FLEXIO\_Type \* `flexioBase`  
*FlexIO module base address.*
- uint32\_t `datPinStartIdx`  
*First data pin (D0) index for flexio\_camera.*
- uint32\_t `pclkPinIdx`  
*Pixel clock pin (PCLK) index for flexio\_camera.*
- uint32\_t `hrefPinIdx`  
*Horizontal sync pin (HREF) index for flexio\_camera.*
- uint32\_t `shifterStartIdx`  
*First shifter index used for flexio\_camera data FIFO.*
- uint32\_t `shifterCount`  
*The count of shifters that are used as flexio\_camera data FIFO.*
- uint32\_t `timerIdx`  
*Timer index used for flexio\_camera in FlexIO.*

##### Field Documentation

(1) `FLEXIO_Type* FLEXIO_CAMERA_Type::flexioBase`

(2) `uint32_t FLEXIO_CAMERA_Type::datPinStartIdx`

Then the successive following `FLEXIO_CAMERA_DATA_WIDTH-1` pins are used as D1-D7.

(3) `uint32_t FLEXIO_CAMERA_Type::pclkPinIdx`

(4) `uint32_t FLEXIO_CAMERA_Type::hrefPinIdx`

(5) `uint32_t FLEXIO_CAMERA_Type::shifterStartIdx`

(6) `uint32_t FLEXIO_CAMERA_Type::shifterCount`

(7) `uint32_t FLEXIO_CAMERA_Type::timerIdx`

#### 23.3.3.2 struct flexio\_camera\_config\_t

##### Data Fields

- bool `enablecamera`  
*Enable/disable FlexIO Camera TX & RX.*

- bool `enableInDoze`  
*Enable/disable FlexIO operation in doze mode.*
- bool `enableInDebug`  
*Enable/disable FlexIO operation in debug mode.*
- bool `enableFastAccess`  
*Enable/disable fast access to FlexIO registers,  
fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*

### Field Documentation

(1) bool `flexio_camera_config_t::enablecamera`

(2) bool `flexio_camera_config_t::enableFastAccess`

#### 23.3.3.3 struct `flexio_camera_transfer_t`

### Data Fields

- uint32\_t `dataAddress`  
*Transfer buffer.*
- uint32\_t `dataNum`  
*Transfer num.*

### 23.3.4 Macro Definition Documentation

23.3.4.1 #define `FSL_FLEXIO_CAMERA_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))`

23.3.4.2 #define `FLEXIO_CAMERA_PARALLEL_DATA_WIDTH (8U)`

### 23.3.5 Enumeration Type Documentation

#### 23.3.5.1 anonymous enum

Enumerator

*`kStatus_FLEXIO_CAMERA_RxBusy`* Receiver is busy.  
*`kStatus_FLEXIO_CAMERA_RxIdle`* Camera receiver is idle.

#### 23.3.5.2 enum `_flexio_camera_status_flags`

Enumerator

*`kFLEXIO_CAMERA_RxDataRegFullFlag`* Receive buffer full flag.  
*`kFLEXIO_CAMERA_RxErrorFlag`* Receive buffer error flag.

### 23.3.6 Function Documentation

23.3.6.1 void FLEXIO\_CAMERA\_Init ( FLEXIO\_CAMERA\_Type \* *base*, const flexio\_camera\_config\_t \* *config* )

Parameters

|               |                                                             |
|---------------|-------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_CAMERA_Type</a> structure     |
| <i>config</i> | Pointer to <a href="#">flexio_camera_config_t</a> structure |

### 23.3.6.2 void FLEXIO\_CAMERA\_Deinit ( FLEXIO\_CAMERA\_Type \* *base* )

Note

After calling this API, call [FLEXIO\\_CAMERA\\_Init](#) to use the FlexIO Camera module.

Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_CAMERA_Type</a> structure |
|-------------|---------------------------------------------------------|

### 23.3.6.3 void FLEXIO\_CAMERA\_GetDefaultConfig ( flexio\_camera\_config\_t \* *config* )

The configuration can be used directly for calling the [FLEXIO\\_CAMERA\\_Init\(\)](#). Example:

```
flexio_camera_config_t config;
FLEXIO_CAMERA_GetDefaultConfig(&userConfig);
```

Parameters

|               |                                                                 |
|---------------|-----------------------------------------------------------------|
| <i>config</i> | Pointer to the <a href="#">flexio_camera_config_t</a> structure |
|---------------|-----------------------------------------------------------------|

### 23.3.6.4 static void FLEXIO\_CAMERA\_Enable ( FLEXIO\_CAMERA\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                                   |
|---------------|---------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_CAMERA_Type</a> |
| <i>enable</i> | True to enable, false does not have any effect.   |

### 23.3.6.5 uint32\_t FLEXIO\_CAMERA\_GetStatusFlags ( FLEXIO\_CAMERA\_Type \* *base* )

## Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_CAMERA_Type</a> structure |
|-------------|---------------------------------------------------------|

## Returns

FlexIO shifter status flags

- FLEXIO\_SHIFTSTAT\_SSF\_MASK
- 0

### 23.3.6.6 void FLEXIO\_CAMERA\_ClearStatusFlags ( FLEXIO\_CAMERA\_Type \* *base*, uint32\_t *mask* )

## Parameters

|             |                                                                                                                                                                                                      |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to the device.                                                                                                                                                                               |
| <i>mask</i> | status flag The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• kFLEXIO_CAMERA_RxDataRegFullFlag</li> <li>• kFLEXIO_CAMERA_RxErrorFlag</li> </ul> |

### 23.3.6.7 void FLEXIO\_CAMERA\_EnableInterrupt ( FLEXIO\_CAMERA\_Type \* *base* )

## Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | Pointer to the device. |
|-------------|------------------------|

### 23.3.6.8 void FLEXIO\_CAMERA\_DisableInterrupt ( FLEXIO\_CAMERA\_Type \* *base* )

## Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | Pointer to the device. |
|-------------|------------------------|

### 23.3.6.9 static void FLEXIO\_CAMERA\_EnableRxDMA ( FLEXIO\_CAMERA\_Type \* *base*, bool *enable* ) [inline], [static]

## Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_CAMERA_Type</a> structure |
| <i>enable</i> | True to enable, false to disable.                       |

The FlexIO Camera mode can't work without the DMA or eDMA support, Usually, it needs at least two DMA or eDMA channels, one for transferring data from Camera, such as 0V7670 to FlexIO buffer, another is for transferring data from FlexIO buffer to LCD.

#### 23.3.6.10 **static uint32\_t FLEXIO\_CAMERA\_GetRxBufferAddress ( FLEXIO\_CAMERA\_Type \* *base* ) [inline], [static]**

## Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | Pointer to the device. |
|-------------|------------------------|

## Returns

data Pointer to the buffer that keeps the data with count of `base->shifterCount` .



## 23.3.7 FlexIO eDMA Camera Driver

### 23.3.7.1 Overview

#### Data Structures

- struct `flexio_camera_edma_handle_t`  
*Camera eDMA handle. [More...](#)*

#### Typedefs

- typedef void(\* `flexio_camera_edma_transfer_callback_t`)(`FLEXIO_CAMERA_Type` \*base, `flexio_camera_edma_handle_t` \*handle, `status_t` status, void \*userData)  
*Camera transfer callback function.*

#### Driver version

- #define `FSL_FLEXIO_CAMERA_EDMA_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 3)`)  
*FlexIO Camera EDMA driver version 2.1.3.*

#### eDMA transactional

- `status_t FLEXIO_CAMERA_TransferCreateHandleEDMA` (`FLEXIO_CAMERA_Type` \*base, `flexio_camera_edma_handle_t` \*handle, `flexio_camera_edma_transfer_callback_t` callback, void \*userData, `edma_handle_t` \*rxEdmaHandle)  
*Initializes the Camera handle, which is used in transactional functions.*
- `status_t FLEXIO_CAMERA_TransferReceiveEDMA` (`FLEXIO_CAMERA_Type` \*base, `flexio_camera_edma_handle_t` \*handle, `flexio_camera_transfer_t` \*xfer)  
*Receives data using eDMA.*
- void `FLEXIO_CAMERA_TransferAbortReceiveEDMA` (`FLEXIO_CAMERA_Type` \*base, `flexio_camera_edma_handle_t` \*handle)  
*Aborts the receive data which used the eDMA.*
- `status_t FLEXIO_CAMERA_TransferGetReceiveCountEDMA` (`FLEXIO_CAMERA_Type` \*base, `flexio_camera_edma_handle_t` \*handle, `size_t` \*count)  
*Gets the remaining bytes to be received.*

### 23.3.7.2 Data Structure Documentation

#### 23.3.7.2.1 struct `flexio_camera_edma_handle`

Forward declaration of the handle typedef.

#### Data Fields

- `flexio_camera_edma_transfer_callback_t` callback

- *Callback function.*
- void \* [userData](#)  
*Camera callback function parameter.*
- size\_t [rxSize](#)  
*Total bytes to be received.*
- [edma\\_handle\\_t](#) \* [rxEdmaHandle](#)  
*The eDMA RX channel used.*
- uint8\_t [nbytes](#)  
*eDMA minor byte transfer count initially configured.*
- volatile uint8\_t [rxState](#)  
*RX transfer state.*

### Field Documentation

- (1) [flexio\\_camera\\_edma\\_transfer\\_callback\\_t](#) [flexio\\_camera\\_edma\\_handle\\_t::callback](#)
- (2) void\* [flexio\\_camera\\_edma\\_handle\\_t::userData](#)
- (3) size\_t [flexio\\_camera\\_edma\\_handle\\_t::rxSize](#)
- (4) [edma\\_handle\\_t](#)\* [flexio\\_camera\\_edma\\_handle\\_t::rxEdmaHandle](#)
- (5) uint8\_t [flexio\\_camera\\_edma\\_handle\\_t::nbytes](#)

### 23.3.7.3 Macro Definition Documentation

23.3.7.3.1 #define FSL\_FLEXIO\_CAMERA\_EDMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 3))

### 23.3.7.4 Typedef Documentation

23.3.7.4.1 typedef void(\* [flexio\\_camera\\_edma\\_transfer\\_callback\\_t](#))([FLEXIO\\_CAMERA\\_Type](#) \*[base](#), [flexio\\_camera\\_edma\\_handle\\_t](#) \*[handle](#), [status\\_t](#) [status](#), void \*[userData](#))

### 23.3.7.5 Function Documentation

23.3.7.5.1 [status\\_t](#) [FLEXIO\\_CAMERA\\_TransferCreateHandleEDMA](#) ( [FLEXIO\\_CAMERA\\_Type](#) \* [base](#), [flexio\\_camera\\_edma\\_handle\\_t](#) \* [handle](#), [flexio\\_camera\\_edma\\_transfer\\_callback\\_t](#) [callback](#), void \* [userData](#), [edma\\_handle\\_t](#) \* [rxEdmaHandle](#) )

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_CAMERA_Type</a> . |
|-------------|-----------------------------------------------------|

|                     |                                                   |
|---------------------|---------------------------------------------------|
| <i>handle</i>       | Pointer to flexio_camera_edma_handle_t structure. |
| <i>callback</i>     | The callback function.                            |
| <i>userData</i>     | The parameter of the callback function.           |
| <i>rxEdmaHandle</i> | User requested DMA handle for RX DMA transfer.    |

Return values

|                           |                                                        |
|---------------------------|--------------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                        |
| <i>kStatus_OutOfRange</i> | The FlexIO Camera eDMA type/handle table out of range. |

### 23.3.7.5.2 status\_t FLEXIO\_CAMERA\_TransferReceiveEDMA ( FLEXIO\_CAMERA\_Type \* base, flexio\_camera\_edma\_handle\_t \* handle, flexio\_camera\_transfer\_t \* xfer )

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

|               |                                                                                |
|---------------|--------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_CAMERA_Type</a> .                            |
| <i>handle</i> | Pointer to the flexio_camera_edma_handle_t structure.                          |
| <i>xfer</i>   | Camera eDMA transfer structure, see <a href="#">flexio_camera_transfer_t</a> . |

Return values

|                               |                              |
|-------------------------------|------------------------------|
| <i>kStatus_Success</i>        | if succeeded, others failed. |
| <i>kStatus_CAMERA_Rx-Busy</i> | Previous transfer on going.  |

### 23.3.7.5.3 void FLEXIO\_CAMERA\_TransferAbortReceiveEDMA ( FLEXIO\_CAMERA\_Type \* base, flexio\_camera\_edma\_handle\_t \* handle )

This function aborts the receive data which used the eDMA.

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_CAMERA_Type</a> . |
|-------------|-----------------------------------------------------|

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>handle</i> | Pointer to the flexio_camera_edma_handle_t structure. |
|---------------|-------------------------------------------------------|

**23.3.7.5.4** `status_t FLEXIO_CAMERA_TransferGetReceiveCountEDMA ( FLEXIO_CAMERA_Type * base, flexio_camera_edma_handle_t * handle, size_t * count )`

This function gets the number of bytes still not received.

Parameters

|               |                                                              |
|---------------|--------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_CAMERA_Type</a> .          |
| <i>handle</i> | Pointer to the flexio_camera_edma_handle_t structure.        |
| <i>count</i>  | Number of bytes sent so far by the non-blocking transaction. |

Return values

|                                |                                 |
|--------------------------------|---------------------------------|
| <i>kStatus_Success</i>         | Succeed get the transfer count. |
| <i>kStatus_InvalidArgument</i> | The count parameter is invalid. |

## 23.4 FlexIO I2C Master Driver

### 23.4.1 Overview

The MCUXpresso SDK provides a peripheral driver for I2C master function using Flexible I/O module of MCUXpresso SDK devices.

The FlexIO I2C master driver includes functional APIs and transactional APIs.

Functional APIs target low level APIs. Functional APIs can be used for the FlexIO I2C master initialization/configuration/operation for the optimization/customization purpose. Using the functional APIs requires the knowledge of the FlexIO I2C master peripheral and how to organize functional APIs to meet the application requirements. The FlexIO I2C master functional operation groups provide the functional APIs set.

Transactional APIs target high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code using the functional APIs or accessing the hardware registers.

Transactional APIs support an asynchronous transfer. This means that the functions [FLEXIO\\_I2C\\_MasterTransferNonBlocking\(\)](#) set up the interrupt non-blocking transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_Success` status.

### 23.4.2 Typical use case

#### 23.4.2.1 FlexIO I2C master transfer using an interrupt method

```
flexio_i2c_master_handle_t g_m_handle;
flexio_i2c_master_config_t masterConfig;
flexio_i2c_master_transfer_t masterXfer;
volatile bool completionFlag = false;
const uint8_t sendData[] = {.....};
FLEXIO_I2C_Type i2cDev;

void FLEXIO_I2C_MasterCallback(FLEXIO_I2C_Type *base, status_t status, void *
 userData)
{
 userData = userData;

 if (kStatus_Success == status)
 {
 completionFlag = true;
 }
}

void main(void)
{
 //...

 FLEXIO_I2C_MasterGetDefaultConfig(&masterConfig);

 FLEXIO_I2C_MasterInit(&i2cDev, &user_config);
 FLEXIO_I2C_MasterTransferCreateHandle(&i2cDev, &g_m_handle,
 FLEXIO_I2C_MasterCallback, NULL);
}
```

```

// Prepares to send.
masterXfer.slaveAddress = g_accel_address[0];
masterXfer.direction = kI2C_Read;
masterXfer.subaddress = &who_am_i_reg;
masterXfer.subaddressSize = 1;
masterXfer.data = &who_am_i_value;
masterXfer.dataSize = 1;
masterXfer.flags = kI2C_TransferDefaultFlag;

// Sends out.
FLEXIO_I2C_MasterTransferNonBlocking(&i2cDev, &g_m_handle, &
 masterXfer);

// Wait for sending is complete.
while (!completionFlag)
{
}

// ...
}

```

## Data Structures

- struct [FLEXIO\\_I2C\\_Type](#)  
Define FlexIO I2C master access structure typedef. [More...](#)
- struct [flexio\\_i2c\\_master\\_config\\_t](#)  
Define FlexIO I2C master user configuration structure. [More...](#)
- struct [flexio\\_i2c\\_master\\_transfer\\_t](#)  
Define FlexIO I2C master transfer structure. [More...](#)
- struct [flexio\\_i2c\\_master\\_handle\\_t](#)  
Define FlexIO I2C master handle structure. [More...](#)

## Macros

- `#define I2C_RETRY_TIMES 0U` /\* Define to zero means keep waiting until the flag is assert/deassert. \*/  
*Retry times for waiting flag.*

## Typedefs

- typedef void(\* [flexio\\_i2c\\_master\\_transfer\\_callback\\_t](#) )(FLEXIO\_I2C\_Type \*base, flexio\_i2c\_master\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*FlexIO I2C master transfer callback typedef.*

## Enumerations

- enum {  
[kStatus\\_FLEXIO\\_I2C\\_Busy](#) = MAKE\_STATUS(kStatusGroup\_FLEXIO\_I2C, 0),  
[kStatus\\_FLEXIO\\_I2C\\_Idle](#) = MAKE\_STATUS(kStatusGroup\_FLEXIO\_I2C, 1),  
[kStatus\\_FLEXIO\\_I2C\\_Nak](#) = MAKE\_STATUS(kStatusGroup\_FLEXIO\_I2C, 2),

- ```
kStatus_FLEXIO_I2C_Timeout = MAKE_STATUS(kStatusGroup_FLEXIO_I2C, 3) }
```
- FlexIO I2C transfer status.*
- enum `_flexio_i2c_master_interrupt` {
 - `kFLEXIO_I2C_TxEmptyInterruptEnable` = 0x1U,
 - `kFLEXIO_I2C_RxFullInterruptEnable` = 0x2U }*Define FlexIO I2C master interrupt mask.*
 - enum `_flexio_i2c_master_status_flags` {
 - `kFLEXIO_I2C_TxEmptyFlag` = 0x1U,
 - `kFLEXIO_I2C_RxFullFlag` = 0x2U,
 - `kFLEXIO_I2C_ReceiveNakFlag` = 0x4U }*Define FlexIO I2C master status mask.*
 - enum `flexio_i2c_direction_t` {
 - `kFLEXIO_I2C_Write` = 0x0U,
 - `kFLEXIO_I2C_Read` = 0x1U }*Direction of master transfer.*

Driver version

- #define `FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION` (`MAKE_VERSION(2, 5, 0)`)

Initialization and deinitialization

- `status_t FLEXIO_I2C_MasterInit` (`FLEXIO_I2C_Type *base`, `flexio_i2c_master_config_t *masterConfig`, `uint32_t srcClock_Hz`)
 - Un-gates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.*
- void `FLEXIO_I2C_MasterDeinit` (`FLEXIO_I2C_Type *base`)
 - De-initializes the FlexIO I2C master peripheral.*
- void `FLEXIO_I2C_MasterGetDefaultConfig` (`flexio_i2c_master_config_t *masterConfig`)
 - Gets the default configuration to configure the FlexIO module.*
- static void `FLEXIO_I2C_MasterEnable` (`FLEXIO_I2C_Type *base`, `bool enable`)
 - Enables/disables the FlexIO module operation.*

Status

- `uint32_t FLEXIO_I2C_MasterGetStatusFlags` (`FLEXIO_I2C_Type *base`)
 - Gets the FlexIO I2C master status flags.*
- void `FLEXIO_I2C_MasterClearStatusFlags` (`FLEXIO_I2C_Type *base`, `uint32_t mask`)
 - Clears the FlexIO I2C master status flags.*

Interrupts

- void `FLEXIO_I2C_MasterEnableInterrupts` (`FLEXIO_I2C_Type *base`, `uint32_t mask`)
 - Enables the FlexIO i2c master interrupt requests.*

- void `FLEXIO_I2C_MasterDisableInterrupts` (`FLEXIO_I2C_Type *base`, `uint32_t mask`)
Disables the FlexIO I2C master interrupt requests.

Bus Operations

- void `FLEXIO_I2C_MasterSetBaudRate` (`FLEXIO_I2C_Type *base`, `uint32_t baudRate_Bps`, `uint32_t srcClock_Hz`)
Sets the FlexIO I2C master transfer baudrate.
- void `FLEXIO_I2C_MasterStart` (`FLEXIO_I2C_Type *base`, `uint8_t address`, `flexio_i2c_direction_t direction`)
Sends START + 7-bit address to the bus.
- void `FLEXIO_I2C_MasterStop` (`FLEXIO_I2C_Type *base`)
Sends the stop signal on the bus.
- void `FLEXIO_I2C_MasterRepeatedStart` (`FLEXIO_I2C_Type *base`)
Sends the repeated start signal on the bus.
- void `FLEXIO_I2C_MasterAbortStop` (`FLEXIO_I2C_Type *base`)
Sends the stop signal when transfer is still on-going.
- void `FLEXIO_I2C_MasterEnableAck` (`FLEXIO_I2C_Type *base`, `bool enable`)
Configures the sent ACK/NAK for the following byte.
- `status_t` `FLEXIO_I2C_MasterSetTransferCount` (`FLEXIO_I2C_Type *base`, `uint16_t count`)
Sets the number of bytes to be transferred from a start signal to a stop signal.
- static void `FLEXIO_I2C_MasterWriteByte` (`FLEXIO_I2C_Type *base`, `uint32_t data`)
Writes one byte of data to the I2C bus.
- static `uint8_t` `FLEXIO_I2C_MasterReadByte` (`FLEXIO_I2C_Type *base`)
Reads one byte of data from the I2C bus.
- `status_t` `FLEXIO_I2C_MasterWriteBlocking` (`FLEXIO_I2C_Type *base`, `const uint8_t *txBuff`, `uint8_t txSize`)
Sends a buffer of data in bytes.
- `status_t` `FLEXIO_I2C_MasterReadBlocking` (`FLEXIO_I2C_Type *base`, `uint8_t *rxBuff`, `uint8_t rxSize`)
Receives a buffer of bytes.
- `status_t` `FLEXIO_I2C_MasterTransferBlocking` (`FLEXIO_I2C_Type *base`, `flexio_i2c_master_transfer_t *xfer`)
Performs a master polling transfer on the I2C bus.

Transactional

- `status_t` `FLEXIO_I2C_MasterTransferCreateHandle` (`FLEXIO_I2C_Type *base`, `flexio_i2c_master_handle_t *handle`, `flexio_i2c_master_transfer_callback_t callback`, `void *userData`)
Initializes the I2C handle which is used in transactional functions.
- `status_t` `FLEXIO_I2C_MasterTransferNonBlocking` (`FLEXIO_I2C_Type *base`, `flexio_i2c_master_handle_t *handle`, `flexio_i2c_master_transfer_t *xfer`)
Performs a master interrupt non-blocking transfer on the I2C bus.
- `status_t` `FLEXIO_I2C_MasterTransferGetCount` (`FLEXIO_I2C_Type *base`, `flexio_i2c_master_handle_t *handle`, `size_t *count`)
Gets the master transfer status during a interrupt non-blocking transfer.

- void `FLEXIO_I2C_MasterTransferAbort` (`FLEXIO_I2C_Type *base`, `flexio_i2c_master_handle_t *handle`)
Aborts an interrupt non-blocking transfer early.
- void `FLEXIO_I2C_MasterTransferHandleIRQ` (`void *i2cType`, `void *i2cHandle`)
Master interrupt handler.

23.4.3 Data Structure Documentation

23.4.3.1 struct FLEXIO_I2C_Type

Data Fields

- `FLEXIO_Type * flexioBase`
FlexIO base pointer.
- `uint8_t SDAPinIndex`
Pin select for I2C SDA.
- `uint8_t SCLPinIndex`
Pin select for I2C SCL.
- `uint8_t shifterIndex` [2]
Shifter index used in FlexIO I2C.
- `uint8_t timerIndex` [3]
Timer index used in FlexIO I2C.
- `uint32_t baudrate`
Master transfer baudrate, used to calculate delay time.

Field Documentation

- (1) `FLEXIO_Type* FLEXIO_I2C_Type::flexioBase`
- (2) `uint8_t FLEXIO_I2C_Type::SDAPinIndex`
- (3) `uint8_t FLEXIO_I2C_Type::SCLPinIndex`
- (4) `uint8_t FLEXIO_I2C_Type::shifterIndex[2]`
- (5) `uint8_t FLEXIO_I2C_Type::timerIndex[3]`
- (6) `uint32_t FLEXIO_I2C_Type::baudrate`

23.4.3.2 struct flexio_i2c_master_config_t

Data Fields

- bool `enableMaster`
Enables the FlexIO I2C peripheral at initialization time.
- bool `enableInDoze`
Enable/disable FlexIO operation in doze mode.
- bool `enableInDebug`
Enable/disable FlexIO operation in debug mode.

- bool [enableFastAccess](#)
Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.
- uint32_t [baudRate_Bps](#)
Baud rate in Bps.

Field Documentation

- (1) bool flexio_i2c_master_config_t::enableMaster
- (2) bool flexio_i2c_master_config_t::enableInDoze
- (3) bool flexio_i2c_master_config_t::enableInDebug
- (4) bool flexio_i2c_master_config_t::enableFastAccess
- (5) uint32_t flexio_i2c_master_config_t::baudRate_Bps

23.4.3.3 struct flexio_i2c_master_transfer_t

Data Fields

- uint32_t [flags](#)
Transfer flag which controls the transfer, reserved for FlexIO I2C.
- uint8_t [slaveAddress](#)
7-bit slave address.
- [flexio_i2c_direction_t](#) [direction](#)
Transfer direction, read or write.
- uint32_t [subaddress](#)
Sub address.
- uint8_t [subaddressSize](#)
Size of command buffer.
- uint8_t volatile * [data](#)
Transfer buffer.
- volatile size_t [dataSize](#)
Transfer size.

Field Documentation

- (1) uint32_t flexio_i2c_master_transfer_t::flags
- (2) uint8_t flexio_i2c_master_transfer_t::slaveAddress
- (3) flexio_i2c_direction_t flexio_i2c_master_transfer_t::direction
- (4) uint32_t flexio_i2c_master_transfer_t::subaddress

Transferred MSB first.

(5) `uint8_t flexio_i2c_master_transfer_t::subaddressSize`

(6) `uint8_t volatile* flexio_i2c_master_transfer_t::data`

(7) `volatile size_t flexio_i2c_master_transfer_t::dataSize`

23.4.3.4 struct `flexio_i2c_master_handle`

FlexIO I2C master handle typedef.

Data Fields

- `flexio_i2c_master_transfer_t transfer`
FlexIO I2C master transfer copy.
- `size_t transferSize`
Total bytes to be transferred.
- `uint8_t state`
Transfer state maintained during transfer.
- `flexio_i2c_master_transfer_callback_t completionCallback`
Callback function called at transfer event.
- `void * userData`
Callback parameter passed to callback function.
- `bool needRestart`
Whether master needs to send re-start signal.

Field Documentation

(1) `flexio_i2c_master_transfer_t flexio_i2c_master_handle_t::transfer`

(2) `size_t flexio_i2c_master_handle_t::transferSize`

(3) `uint8_t flexio_i2c_master_handle_t::state`

(4) `flexio_i2c_master_transfer_callback_t flexio_i2c_master_handle_t::completionCallback`

Callback function called at transfer event.

- (5) void* flexio_i2c_master_handle_t::userData
- (6) bool flexio_i2c_master_handle_t::needRestart

23.4.4 Macro Definition Documentation

- 23.4.4.1 #define I2C_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */

23.4.5 Typedef Documentation

- 23.4.5.1 typedef void(* flexio_i2c_master_transfer_callback_t)(FLEXIO_I2C_Type *base, flexio_i2c_master_handle_t *handle, status_t status, void *userData)

23.4.6 Enumeration Type Documentation

23.4.6.1 anonymous enum

Enumerator

- kStatus_FLEXIO_I2C_Busy* I2C is busy doing transfer.
- kStatus_FLEXIO_I2C_Idle* I2C is busy doing transfer.
- kStatus_FLEXIO_I2C_Nak* NAK received during transfer.
- kStatus_FLEXIO_I2C_Timeout* Timeout polling status flags.

23.4.6.2 enum flexio_i2c_master_interrupt

Enumerator

- kFLEXIO_I2C_TxEmptyInterruptEnable* Tx buffer empty interrupt enable.
- kFLEXIO_I2C_RxFullInterruptEnable* Rx buffer full interrupt enable.

23.4.6.3 enum flexio_i2c_master_status_flags

Enumerator

- kFLEXIO_I2C_TxEmptyFlag* Tx shifter empty flag.
- kFLEXIO_I2C_RxFullFlag* Rx shifter full/Transfer complete flag.
- kFLEXIO_I2C_ReceiveNakFlag* Receive NAK flag.

23.4.6.4 enum flexio_i2c_direction_t

Enumerator

- kFLEXIO_I2C_Write* Master send to slave.
- kFLEXIO_I2C_Read* Master receive from slave.

23.4.7 Function Documentation

23.4.7.1 status_t FLEXIO_I2C_MasterInit (FLEXIO_I2C_Type * *base*, flexio_i2c_master_config_t * *masterConfig*, uint32_t *srcClock_Hz*)

Example

```
FLEXIO_I2C_Type base = {
    .flexioBase = FLEXIO,
    .SDAPinIndex = 0,
    .SCLPinIndex = 1,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_i2c_master_config_t config = {
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 100000
};
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
<i>masterConfig</i>	Pointer to flexio_i2c_master_config_t structure.
<i>srcClock_Hz</i>	FlexIO source clock in Hz.

Return values

<i>kStatus_Success</i>	Initialization successful
<i>kStatus_InvalidArgument</i>	The source clock exceed upper range limitation

23.4.7.2 void FLEXIO_I2C_MasterDeinit (FLEXIO_I2C_Type * *base*)

Calling this API Resets the FlexIO I2C master shifer and timer config, module can't work unless the FLEXIO_I2C_MasterInit is called.

Parameters

<i>base</i>	pointer to FLEXIO_I2C_Type structure.
-------------	---

23.4.7.3 void FLEXIO_I2C_MasterGetDefaultConfig (flexio_i2c_master_config_t * masterConfig)

The configuration can be used directly for calling the [FLEXIO_I2C_MasterInit\(\)](#).

Example:

```
flexio_i2c_master_config_t config;
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

Parameters

<i>masterConfig</i>	Pointer to flexio_i2c_master_config_t structure.
---------------------	--

23.4.7.4 static void FLEXIO_I2C_MasterEnable (FLEXIO_I2C_Type * base, bool enable) [inline], [static]

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
<i>enable</i>	Pass true to enable module, false does not have any effect.

23.4.7.5 uint32_t FLEXIO_I2C_MasterGetStatusFlags (FLEXIO_I2C_Type * base)

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure
-------------	--

Returns

Status flag, use status flag to AND [_flexio_i2c_master_status_flags](#) can get the related status.

23.4.7.6 void FLEXIO_I2C_MasterClearStatusFlags (FLEXIO_I2C_Type * base, uint32_t mask)

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
<i>mask</i>	Status flag. The parameter can be any combination of the following values: <ul style="list-style-type: none"> • kFLEXIO_I2C_RxFullFlag • kFLEXIO_I2C_ReceiveNakFlag

23.4.7.7 void FLEXIO_I2C_MasterEnableInterrupts (FLEXIO_I2C_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
<i>mask</i>	Interrupt source. Currently only one interrupt request source: <ul style="list-style-type: none"> • kFLEXIO_I2C_TransferCompleteInterruptEnable

23.4.7.8 void FLEXIO_I2C_MasterDisableInterrupts (FLEXIO_I2C_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
<i>mask</i>	Interrupt source.

23.4.7.9 void FLEXIO_I2C_MasterSetBaudRate (FLEXIO_I2C_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz*)

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure
<i>baudRate_Bps</i>	the baud rate value in HZ

<i>srcClock_Hz</i>	source clock in HZ
--------------------	--------------------

23.4.7.10 void FLEXIO_I2C_MasterStart (FLEXIO_I2C_Type * *base*, uint8_t *address*, flexio_i2c_direction_t *direction*)

Note

This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the kFLEXIO_I2C_RxFullFlag status is asserted before calling this API.

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
<i>address</i>	7-bit address.
<i>direction</i>	transfer direction. This parameter is one of the values in flexio_i2c_direction_t: <ul style="list-style-type: none"> • kFLEXIO_I2C_Write: Transmit • kFLEXIO_I2C_Read: Receive

23.4.7.11 void FLEXIO_I2C_MasterStop (FLEXIO_I2C_Type * *base*)

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
-------------	---

23.4.7.12 void FLEXIO_I2C_MasterRepeatedStart (FLEXIO_I2C_Type * *base*)

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
-------------	---

23.4.7.13 void FLEXIO_I2C_MasterAbortStop (FLEXIO_I2C_Type * *base*)

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
-------------	---

23.4.7.14 void FLEXIO_I2C_MasterEnableAck (FLEXIO_I2C_Type * *base*, bool *enable*)

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
<i>enable</i>	True to configure send ACK, false configure to send NAK.

23.4.7.15 status_t FLEXIO_I2C_MasterSetTransferCount (FLEXIO_I2C_Type * *base*, uint16_t *count*)

Note

Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
<i>count</i>	Number of bytes need to be transferred from a start signal to a re-start/stop signal

Return values

<i>kStatus_Success</i>	Successfully configured the count.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.

23.4.7.16 static void FLEXIO_I2C_MasterWriteByte (FLEXIO_I2C_Type * *base*, uint32_t *data*) [inline], [static]

Note

This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
<i>data</i>	a byte of data.

23.4.7.17 static uint8_t FLEXIO_I2C_MasterReadByte (FLEXIO_I2C_Type * *base*) [inline], [static]

Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
-------------	---

Returns

data byte read.

23.4.7.18 status_t FLEXIO_I2C_MasterWriteBlocking (FLEXIO_I2C_Type * *base*, const uint8_t * *txBuff*, uint8_t *txSize*)

Note

This function blocks via polling until all bytes have been sent.

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
<i>txBuff</i>	The data bytes to send.
<i>txSize</i>	The number of data bytes to send.

Return values

<i>kStatus_Success</i>	Successfully write data.
<i>kStatus_FLEXIO_I2C_-Nak</i>	Receive NAK during writing data.
<i>kStatus_FLEXIO_I2C_-Timeout</i>	Timeout polling status flags.

23.4.7.19 `status_t FLEXIO_I2C_MasterReadBlocking (FLEXIO_I2C_Type * base, uint8_t * rxBuff, uint8_t rxSize)`

Note

This function blocks via polling until all bytes have been received.

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
<i>rxBuff</i>	The buffer to store the received bytes.
<i>rxSize</i>	The number of data bytes to be received.

Return values

<i>kStatus_Success</i>	Successfully read data.
<i>kStatus_FLEXIO_I2C_-Timeout</i>	Timeout polling status flags.

23.4.7.20 `status_t FLEXIO_I2C_MasterTransferBlocking (FLEXIO_I2C_Type * base, flexio_i2c_master_transfer_t * xfer)`

Note

The API does not return until the transfer succeeds or fails due to receiving NAK.

Parameters

<i>base</i>	pointer to FLEXIO_I2C_Type structure.
-------------	---

<i>xfer</i>	pointer to flexio_i2c_master_transfer_t structure.
-------------	--

Returns

status of `status_t`.

23.4.7.21 `status_t FLEXIO_I2C_MasterTransferCreateHandle (FLEXIO_I2C_Type * base, flexio_i2c_master_handle_t * handle, flexio_i2c_master_transfer_callback_t callback, void * userData)`

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
<i>handle</i>	Pointer to <code>flexio_i2c_master_handle_t</code> structure to store the transfer state.
<i>callback</i>	Pointer to user callback function.
<i>userData</i>	User param passed to the callback function.

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_OutOfRange</i>	The FlexIO type/handle/isr table out of range.

23.4.7.22 `status_t FLEXIO_I2C_MasterTransferNonBlocking (FLEXIO_I2C_Type * base, flexio_i2c_master_handle_t * handle, flexio_i2c_master_transfer_t * xfer)`

Note

The API returns immediately after the transfer initiates. Call `FLEXIO_I2C_MasterTransferGetCount` to poll the transfer status to check whether the transfer is finished. If the return status is not `kStatus_FLEXIO_I2C_Busy`, the transfer is finished.

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure
<i>handle</i>	Pointer to <code>flexio_i2c_master_handle_t</code> structure which stores the transfer state

<i>xfer</i>	pointer to flexio_i2c_master_transfer_t structure
-------------	---

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_FLEXIO_I2C_Busy</i>	FlexIO I2C is not idle, is running another transfer.

23.4.7.23 **status_t FLEXIO_I2C_MasterTransferGetCount (FLEXIO_I2C_Type * *base*, flexio_i2c_master_handle_t * *handle*, size_t * *count*)**

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
<i>handle</i>	Pointer to flexio_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_NoTransferInProgress</i>	There is not a non-blocking transaction currently in progress.
<i>kStatus_Success</i>	Successfully return the count.

23.4.7.24 **void FLEXIO_I2C_MasterTransferAbort (FLEXIO_I2C_Type * *base*, flexio_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>	Pointer to FLEXIO_I2C_Type structure
-------------	--

<i>handle</i>	Pointer to flexio_i2c_master_handle_t structure which stores the transfer state
---------------	---

23.4.7.25 void FLEXIO_I2C_MasterTransferHandleIRQ (void * *i2cType*, void * *i2cHandle*)

Parameters

<i>i2cType</i>	Pointer to FLEXIO_I2C_Type structure
<i>i2cHandle</i>	Pointer to flexio_i2c_master_transfer_t structure

23.5 FlexIO I2S Driver

23.5.1 Overview

The MCUXpresso SDK provides a peripheral driver for I2S function using Flexible I/O module of MCU-Xpresso SDK devices.

The FlexIO I2S driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs.

Functional APIs can be used for FlexIO I2S initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FlexIO I2S peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. FlexIO I2S functional operation groups provide the functional APIs set.

Transactional APIs are transaction target high level APIs. The transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs can satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the `sai_handle_t` as the first parameter. Initialize the handle by calling the `FlexIO_I2S_TransferTxCreateHandle()` or `FlexIO_I2S_TransferRxCreateHandle()` API.

Transactional APIs support asynchronous transfer. This means that the functions [FLEXIO_I2S_TransferSendNonBlocking\(\)](#) and [FLEXIO_I2S_TransferReceiveNonBlocking\(\)](#) set up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_FLEXIO_I2S_TxIdle` and `kStatus_FLEXIO_I2S_RxIdle` status.

23.5.2 Typical use case

23.5.2.1 FlexIO I2S send/receive using an interrupt method

```
sai_handle_t g_saiTxHandle;
sai_config_t user_config;
sai_transfer_t sendXfer;
volatile bool txFinished;
volatile bool rxFinished;
const uint8_t sendData[] = [.....];

void FLEXIO_I2S_UserCallback(sai_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_I2S_TxIdle == status)
    {
        txFinished = true;
    }
}

void main(void)
{
    //...

    FLEXIO_I2S_TxGetDefaultConfig(&user_config);
```

```

FLEXIO_I2S_TxInit(FLEXIO I2S0, &user_config);
FLEXIO_I2S_TransferTxCreateHandle(FLEXIO I2S0, &g_saiHandle,
    FLEXIO_I2S_UserCallback, NULL);

//Configures the SAI format.
FLEXIO_I2S_TransferTxSetTransferFormat(FLEXIO I2S0, &g_saiHandle, mclkSource, mclk);

// Prepares to send.
sendXfer.data = sendData
sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
txFinished = false;

// Sends out.
FLEXIO_I2S_TransferSendNonBlocking(FLEXIO I2S0, &g_saiHandle, &
    sendXfer);

// Waiting to send is finished.
while (!txFinished)
{
}

// ...
}

```

23.5.2.2 FLEXIO_I2S send/receive using a DMA method

```

sai_handle_t g_saiHandle;
dma_handle_t g_saiTxDmaHandle;
dma_handle_t g_saiRxDmaHandle;
sai_config_t user_config;
sai_transfer_t sendXfer;
volatile bool txFinished;
uint8_t sendData[] = ...;

void FLEXIO_I2S_UserCallback(sai_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_I2S_TxIdle == status)
    {
        txFinished = true;
    }
}

void main(void)
{
    //...

    FLEXIO_I2S_TxGetDefaultConfig(&user_config);
    FLEXIO_I2S_TxInit(FLEXIO I2S0, &user_config);

    // Sets up the DMA.
    DMAMUX_Init(DMAMUX0);
    DMAMUX_SetSource(DMAMUX0, FLEXIO_I2S_TX_DMA_CHANNEL, FLEXIO_I2S_TX_DMA_REQUEST);
    DMAMUX_EnableChannel(DMAMUX0, FLEXIO_I2S_TX_DMA_CHANNEL);

    DMA_Init(DMA0);

    /* Creates the DMA handle. */
    DMA_TransferTxCreateHandle(&g_saiTxDmaHandle, DMA0, FLEXIO_I2S_TX_DMA_CHANNEL);

    FLEXIO_I2S_TransferTxCreateHandleDMA(FLEXIO I2S0, &g_saiTxDmaHandle, FLEXIO_I2S_UserCallback, NULL);

    // Prepares to send.
    sendXfer.data = sendData

```



```

sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
txFinished = false;

// Sends out.
FLEXIO_I2S_TransferSendDMA(&g_saiHandle, &sendXfer);

// Waiting to send is finished.
while (!txFinished)
{
}

// ...
}

```

Modules

- [FlexIO eDMA I2S Driver](#)

Data Structures

- struct [FLEXIO_I2S_Type](#)
Define FlexIO I2S access structure typedef. [More...](#)
- struct [flexio_i2s_config_t](#)
FlexIO I2S configure structure. [More...](#)
- struct [flexio_i2s_format_t](#)
FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx. [More...](#)
- struct [flexio_i2s_transfer_t](#)
Define FlexIO I2S transfer structure. [More...](#)
- struct [flexio_i2s_handle_t](#)
Define FlexIO I2S handle structure. [More...](#)

Macros

- `#define I2S_RETRY_TIMES 0U` /* Define to zero means keep waiting until the flag is assert/deassert. */
Retry times for waiting flag.
- `#define FLEXIO_I2S_XFER_QUEUE_SIZE (4U)`
FlexIO I2S transfer queue size, user can refine it according to use case.

Typedefs

- typedef void(* [flexio_i2s_callback_t](#))(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, [status_t](#) status, void *userData)
FlexIO I2S xfer callback prototype.

Enumerations

- enum {
 - kStatus_FLEXIO_I2S_Idle = MAKE_STATUS(kStatusGroup_FLEXIO_I2S, 0),
 - kStatus_FLEXIO_I2S_TxBusy = MAKE_STATUS(kStatusGroup_FLEXIO_I2S, 1),
 - kStatus_FLEXIO_I2S_RxBusy = MAKE_STATUS(kStatusGroup_FLEXIO_I2S, 2),
 - kStatus_FLEXIO_I2S_Error = MAKE_STATUS(kStatusGroup_FLEXIO_I2S, 3),
 - kStatus_FLEXIO_I2S_QueueFull = MAKE_STATUS(kStatusGroup_FLEXIO_I2S, 4),
 - kStatus_FLEXIO_I2S_Timeout }

FlexIO I2S transfer status.
- enum flexio_i2s_master_slave_t {
 - kFLEXIO_I2S_Master = 0x0U,
 - kFLEXIO_I2S_Slave = 0x1U }

Master or slave mode.
- enum {
 - kFLEXIO_I2S_TxDataRegEmptyInterruptEnable = 0x1U,
 - kFLEXIO_I2S_RxDataRegFullInterruptEnable = 0x2U }

_flexio_i2s_interrupt_enable Define FlexIO FlexIO I2S interrupt mask.
- enum {
 - kFLEXIO_I2S_TxDataRegEmptyFlag = 0x1U,
 - kFLEXIO_I2S_RxDataRegFullFlag = 0x2U }

_flexio_i2s_status_flags Define FlexIO FlexIO I2S status mask.
- enum flexio_i2s_sample_rate_t {
 - kFLEXIO_I2S_SampleRate8KHz = 8000U,
 - kFLEXIO_I2S_SampleRate11025Hz = 11025U,
 - kFLEXIO_I2S_SampleRate12KHz = 12000U,
 - kFLEXIO_I2S_SampleRate16KHz = 16000U,
 - kFLEXIO_I2S_SampleRate22050Hz = 22050U,
 - kFLEXIO_I2S_SampleRate24KHz = 24000U,
 - kFLEXIO_I2S_SampleRate32KHz = 32000U,
 - kFLEXIO_I2S_SampleRate44100Hz = 44100U,
 - kFLEXIO_I2S_SampleRate48KHz = 48000U,
 - kFLEXIO_I2S_SampleRate96KHz = 96000U }

Audio sample rate.
- enum flexio_i2s_word_width_t {
 - kFLEXIO_I2S_WordWidth8bits = 8U,
 - kFLEXIO_I2S_WordWidth16bits = 16U,
 - kFLEXIO_I2S_WordWidth24bits = 24U,
 - kFLEXIO_I2S_WordWidth32bits = 32U }

Audio word width.

Driver version

- #define FSL_FLEXIO_I2S_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))

FlexIO I2S driver version 2.2.0.

Initialization and deinitialization

- void `FLEXIO_I2S_Init` (`FLEXIO_I2S_Type *base`, const `flexio_i2s_config_t *config`)
Initializes the FlexIO I2S.
- void `FLEXIO_I2S_GetDefaultConfig` (`flexio_i2s_config_t *config`)
Sets the FlexIO I2S configuration structure to default values.
- void `FLEXIO_I2S_Deinit` (`FLEXIO_I2S_Type *base`)
De-initializes the FlexIO I2S.
- static void `FLEXIO_I2S_Enable` (`FLEXIO_I2S_Type *base`, bool enable)
Enables/disables the FlexIO I2S module operation.

Status

- uint32_t `FLEXIO_I2S_GetStatusFlags` (`FLEXIO_I2S_Type *base`)
Gets the FlexIO I2S status flags.

Interrupts

- void `FLEXIO_I2S_EnableInterrupts` (`FLEXIO_I2S_Type *base`, uint32_t mask)
Enables the FlexIO I2S interrupt.
- void `FLEXIO_I2S_DisableInterrupts` (`FLEXIO_I2S_Type *base`, uint32_t mask)
Disables the FlexIO I2S interrupt.

DMA Control

- static void `FLEXIO_I2S_TxEnableDMA` (`FLEXIO_I2S_Type *base`, bool enable)
Enables/disables the FlexIO I2S Tx DMA requests.
- static void `FLEXIO_I2S_RxEnableDMA` (`FLEXIO_I2S_Type *base`, bool enable)
Enables/disables the FlexIO I2S Rx DMA requests.
- static uint32_t `FLEXIO_I2S_TxGetDataRegisterAddress` (`FLEXIO_I2S_Type *base`)
Gets the FlexIO I2S send data register address.
- static uint32_t `FLEXIO_I2S_RxGetDataRegisterAddress` (`FLEXIO_I2S_Type *base`)
Gets the FlexIO I2S receive data register address.

Bus Operations

- void `FLEXIO_I2S_MasterSetFormat` (`FLEXIO_I2S_Type *base`, `flexio_i2s_format_t *format`, uint32_t srcClock_Hz)
Configures the FlexIO I2S audio format in master mode.
- void `FLEXIO_I2S_SlaveSetFormat` (`FLEXIO_I2S_Type *base`, `flexio_i2s_format_t *format`)
Configures the FlexIO I2S audio format in slave mode.
- `status_t` `FLEXIO_I2S_WriteBlocking` (`FLEXIO_I2S_Type *base`, uint8_t bitWidth, uint8_t *txData, size_t size)
Sends data using a blocking method.
- static void `FLEXIO_I2S_WriteData` (`FLEXIO_I2S_Type *base`, uint8_t bitWidth, uint32_t data)

Writes data into a data register.

- `status_t FLEXIO_I2S_ReadBlocking` (`FLEXIO_I2S_Type *base`, `uint8_t bitWidth`, `uint8_t *rxData`, `size_t size`)

Receives a piece of data using a blocking method.

- `static uint32_t FLEXIO_I2S_ReadData` (`FLEXIO_I2S_Type *base`)

Reads a data from the data register.

Transactional

- `void FLEXIO_I2S_TransferTxCreateHandle` (`FLEXIO_I2S_Type *base`, `flexio_i2s_handle_t *handle`, `flexio_i2s_callback_t callback`, `void *userData`)

Initializes the FlexIO I2S handle.

- `void FLEXIO_I2S_TransferSetFormat` (`FLEXIO_I2S_Type *base`, `flexio_i2s_handle_t *handle`, `flexio_i2s_format_t *format`, `uint32_t srcClock_Hz`)

Configures the FlexIO I2S audio format.

- `void FLEXIO_I2S_TransferRxCreateHandle` (`FLEXIO_I2S_Type *base`, `flexio_i2s_handle_t *handle`, `flexio_i2s_callback_t callback`, `void *userData`)

Initializes the FlexIO I2S receive handle.

- `status_t FLEXIO_I2S_TransferSendNonBlocking` (`FLEXIO_I2S_Type *base`, `flexio_i2s_handle_t *handle`, `flexio_i2s_transfer_t *xfer`)

Performs an interrupt non-blocking send transfer on FlexIO I2S.

- `status_t FLEXIO_I2S_TransferReceiveNonBlocking` (`FLEXIO_I2S_Type *base`, `flexio_i2s_handle_t *handle`, `flexio_i2s_transfer_t *xfer`)

Performs an interrupt non-blocking receive transfer on FlexIO I2S.

- `void FLEXIO_I2S_TransferAbortSend` (`FLEXIO_I2S_Type *base`, `flexio_i2s_handle_t *handle`)

Aborts the current send.

- `void FLEXIO_I2S_TransferAbortReceive` (`FLEXIO_I2S_Type *base`, `flexio_i2s_handle_t *handle`)

Aborts the current receive.

- `status_t FLEXIO_I2S_TransferGetSendCount` (`FLEXIO_I2S_Type *base`, `flexio_i2s_handle_t *handle`, `size_t *count`)

Gets the remaining bytes to be sent.

- `status_t FLEXIO_I2S_TransferGetReceiveCount` (`FLEXIO_I2S_Type *base`, `flexio_i2s_handle_t *handle`, `size_t *count`)

Gets the remaining bytes to be received.

- `void FLEXIO_I2S_TransferTxHandleIRQ` (`void *i2sBase`, `void *i2sHandle`)

Tx interrupt handler.

- `void FLEXIO_I2S_TransferRxHandleIRQ` (`void *i2sBase`, `void *i2sHandle`)

Rx interrupt handler.

23.5.3 Data Structure Documentation

23.5.3.1 struct FLEXIO_I2S_Type

Data Fields

- `FLEXIO_Type * flexioBase`

FlexIO base pointer.

- `uint8_t txPinIndex`
Tx data pin index in FlexIO pins.
- `uint8_t rxPinIndex`
Rx data pin index.
- `uint8_t bclkPinIndex`
Bit clock pin index.
- `uint8_t fsPinIndex`
Frame sync pin index.
- `uint8_t txShifterIndex`
Tx data shifter index.
- `uint8_t rxShifterIndex`
Rx data shifter index.
- `uint8_t bclkTimerIndex`
Bit clock timer index.
- `uint8_t fsTimerIndex`
Frame sync timer index.

23.5.3.2 struct flexio_i2s_config_t

Data Fields

- `bool enableI2S`
Enable FlexIO I2S.
- `flexio_i2s_master_slave_t masterSlave`
Master or slave.
- `flexio_pin_polarity_t txPinPolarity`
Tx data pin polarity, active high or low.
- `flexio_pin_polarity_t rxPinPolarity`
Rx data pin polarity.
- `flexio_pin_polarity_t bclkPinPolarity`
Bit clock pin polarity.
- `flexio_pin_polarity_t fsPinPolarity`
Frame sync pin polarity.
- `flexio_shifter_timer_polarity_t txTimerPolarity`
Tx data valid on bclk rising or falling edge.
- `flexio_shifter_timer_polarity_t rxTimerPolarity`
Rx data valid on bclk rising or falling edge.

23.5.3.3 struct flexio_i2s_format_t

Data Fields

- `uint8_t bitWidth`
Bit width of audio data, always 8/16/24/32 bits.
- `uint32_t sampleRate_Hz`
Sample rate of the audio data.

23.5.3.4 struct flexio_i2s_transfer_t

Data Fields

- uint8_t * [data](#)
Data buffer start pointer.
- size_t [dataSize](#)
Bytes to be transferred.

Field Documentation

(1) [size_t flexio_i2s_transfer_t::dataSize](#)

23.5.3.5 struct _flexio_i2s_handle

Data Fields

- uint32_t [state](#)
Internal state.
- [flexio_i2s_callback_t](#) [callback](#)
Callback function called at transfer event.
- void * [userData](#)
Callback parameter passed to callback function.
- uint8_t [bitWidth](#)
Bit width for transfer, 8/16/24/32bits.
- [flexio_i2s_transfer_t](#) [queue](#) [FLEXIO_I2S_XFER_QUEUE_SIZE]
Transfer queue storing queued transfer.
- size_t [transferSize](#) [FLEXIO_I2S_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.

23.5.4 Macro Definition Documentation

23.5.4.1 `#define FSL_FLEXIO_I2S_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))`

23.5.4.2 `#define I2S_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`

23.5.4.3 `#define FLEXIO_I2S_XFER_QUEUE_SIZE (4U)`

23.5.5 Enumeration Type Documentation

23.5.5.1 anonymous enum

Enumerator

kStatus_FLEXIO_I2S_Idle FlexIO I2S is in idle state.

kStatus_FLEXIO_I2S_TxBusy FlexIO I2S Tx is busy.

kStatus_FLEXIO_I2S_RxBusy FlexIO I2S Rx is busy.

kStatus_FLEXIO_I2S_Error FlexIO I2S error occurred.

kStatus_FLEXIO_I2S_QueueFull FlexIO I2S transfer queue is full.

kStatus_FLEXIO_I2S_Timeout FlexIO I2S timeout polling status flags.

23.5.5.2 enum flexio_i2s_master_slave_t

Enumerator

kFLEXIO_I2S_Master Master mode.

kFLEXIO_I2S_Slave Slave mode.

23.5.5.3 anonymous enum

Enumerator

kFLEXIO_I2S_TxDataRegEmptyInterruptEnable Transmit buffer empty interrupt enable.

kFLEXIO_I2S_RxDataRegFullInterruptEnable Receive buffer full interrupt enable.

23.5.5.4 anonymous enum

Enumerator

kFLEXIO_I2S_TxDataRegEmptyFlag Transmit buffer empty flag.

kFLEXIO_I2S_RxDataRegFullFlag Receive buffer full flag.

23.5.5.5 enum flexio_i2s_sample_rate_t

Enumerator

kFLEXIO_I2S_SampleRate8KHz Sample rate 8000Hz.
kFLEXIO_I2S_SampleRate11025Hz Sample rate 11025Hz.
kFLEXIO_I2S_SampleRate12KHz Sample rate 12000Hz.
kFLEXIO_I2S_SampleRate16KHz Sample rate 16000Hz.
kFLEXIO_I2S_SampleRate22050Hz Sample rate 22050Hz.
kFLEXIO_I2S_SampleRate24KHz Sample rate 24000Hz.
kFLEXIO_I2S_SampleRate32KHz Sample rate 32000Hz.
kFLEXIO_I2S_SampleRate44100Hz Sample rate 44100Hz.
kFLEXIO_I2S_SampleRate48KHz Sample rate 48000Hz.
kFLEXIO_I2S_SampleRate96KHz Sample rate 96000Hz.

23.5.5.6 enum flexio_i2s_word_width_t

Enumerator

kFLEXIO_I2S_WordWidth8bits Audio data width 8 bits.
kFLEXIO_I2S_WordWidth16bits Audio data width 16 bits.
kFLEXIO_I2S_WordWidth24bits Audio data width 24 bits.
kFLEXIO_I2S_WordWidth32bits Audio data width 32 bits.

23.5.6 Function Documentation

23.5.6.1 void FLEXIO_I2S_Init (FLEXIO_I2S_Type * *base*, const flexio_i2s_config_t * *config*)

This API configures FlexIO pins and shifter to I2S and configures the FlexIO I2S with a configuration structure. The configuration structure can be filled by the user, or be set with default values by [FLEXIO_I2S_GetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the FlexIO I2S driver. Otherwise, any access to the FlexIO I2S module can cause hard fault because the clock is not enabled.

Parameters

<i>base</i>	FlexIO I2S base pointer
<i>config</i>	FlexIO I2S configure structure.

23.5.6.2 void FLEXIO_I2S_GetDefaultConfig (flexio_i2s_config_t * config)

The purpose of this API is to get the configuration structure initialized for use in [FLEXIO_I2S_Init\(\)](#). Users may use the initialized structure unchanged in [FLEXIO_I2S_Init\(\)](#) or modify some fields of the structure before calling [FLEXIO_I2S_Init\(\)](#).

Parameters

<i>config</i>	pointer to master configuration structure
---------------	---

23.5.6.3 void FLEXIO_I2S_Deinit (FLEXIO_I2S_Type * base)

Calling this API resets the FlexIO I2S shifter and timer config. After calling this API, call the [FLEXIO_I2S_Init](#) to use the FlexIO I2S module.

Parameters

<i>base</i>	FlexIO I2S base pointer
-------------	-------------------------

23.5.6.4 static void FLEXIO_I2S_Enable (FLEXIO_I2S_Type * base, bool enable) [inline], [static]

Parameters

<i>base</i>	Pointer to FLEXIO_I2S_Type
<i>enable</i>	True to enable, false dose not have any effect.

23.5.6.5 uint32_t FLEXIO_I2S_GetStatusFlags (FLEXIO_I2S_Type * base)

Parameters

<i>base</i>	Pointer to FLEXIO_I2S_Type structure
-------------	--

Returns

Status flag, which are ORed by the enumerators in the `_flexio_i2s_status_flags`.

23.5.6.6 void FLEXIO_I2S_EnableInterrupts (FLEXIO_I2S_Type * *base*, uint32_t *mask*)

This function enables the FlexIO UART interrupt.

Parameters

<i>base</i>	Pointer to FLEXIO_I2S_Type structure
<i>mask</i>	interrupt source

23.5.6.7 void FLEXIO_I2S_DisableInterrupts (FLEXIO_I2S_Type * *base*, uint32_t *mask*)

This function enables the FlexIO UART interrupt.

Parameters

<i>base</i>	pointer to FLEXIO_I2S_Type structure
<i>mask</i>	interrupt source

23.5.6.8 static void FLEXIO_I2S_TxEnableDMA (FLEXIO_I2S_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	FlexIO I2S base pointer
<i>enable</i>	True means enable DMA, false means disable DMA.

23.5.6.9 static void FLEXIO_I2S_RxEnableDMA (FLEXIO_I2S_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	FlexIO I2S base pointer
<i>enable</i>	True means enable DMA, false means disable DMA.

23.5.6.10 **static uint32_t FLEXIO_I2S_TxGetDataRegisterAddress (FLEXIO_I2S_Type * *base*) [inline], [static]**

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

<i>base</i>	Pointer to FLEXIO_I2S_Type structure
-------------	--

Returns

FlexIO i2s send data register address.

23.5.6.11 **static uint32_t FLEXIO_I2S_RxGetDataRegisterAddress (FLEXIO_I2S_Type * *base*) [inline], [static]**

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

<i>base</i>	Pointer to FLEXIO_I2S_Type structure
-------------	--

Returns

FlexIO i2s receive data register address.

23.5.6.12 **void FLEXIO_I2S_MasterSetFormat (FLEXIO_I2S_Type * *base*, *flexio_i2s_format_t* * *format*, uint32_t *srcClock_Hz*)**

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

<i>base</i>	Pointer to FLEXIO_I2S_Type structure
<i>format</i>	Pointer to FlexIO I2S audio data format structure.
<i>srcClock_Hz</i>	I2S master clock source frequency in Hz.

23.5.6.13 void FLEXIO_I2S_SlaveSetFormat (FLEXIO_I2S_Type * *base*, flexio_i2s_format_t * *format*)

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

<i>base</i>	Pointer to FLEXIO_I2S_Type structure
<i>format</i>	Pointer to FlexIO I2S audio data format structure.

23.5.6.14 status_t FLEXIO_I2S_WriteBlocking (FLEXIO_I2S_Type * *base*, uint8_t *bitWidth*, uint8_t * *txData*, size_t *size*)

Note

This function blocks via polling until data is ready to be sent.

Parameters

<i>base</i>	FlexIO I2S base pointer.
<i>bitWidth</i>	How many bits in a audio word, usually 8/16/24/32 bits.
<i>txData</i>	Pointer to the data to be written.
<i>size</i>	Bytes to be written.

Return values

<i>kStatus_Success</i>	Successfully write data.
<i>kStatus_FLEXIO_I2C_Timeout</i>	Timeout polling status flags.

23.5.6.15 static void FLEXIO_I2S_WriteData (FLEXIO_I2S_Type * *base*, uint8_t *bitWidth*, uint32_t *data*) [inline], [static]

Parameters

<i>base</i>	FlexIO I2S base pointer.
<i>bitWidth</i>	How many bits in a audio word, usually 8/16/24/32 bits.
<i>data</i>	Data to be written.

23.5.6.16 `status_t FLEXIO_I2S_ReadBlocking (FLEXIO_I2S_Type * base, uint8_t bitWidth, uint8_t * rxData, size_t size)`

Note

This function blocks via polling until data is ready to be sent.

Parameters

<i>base</i>	FlexIO I2S base pointer
<i>bitWidth</i>	How many bits in a audio word, usually 8/16/24/32 bits.
<i>rxData</i>	Pointer to the data to be read.
<i>size</i>	Bytes to be read.

Return values

<i>kStatus_Success</i>	Successfully read data.
<i>kStatus_FLEXIO_I2C_-Timeout</i>	Timeout polling status flags.

23.5.6.17 `static uint32_t FLEXIO_I2S_ReadData (FLEXIO_I2S_Type * base)` `[inline], [static]`

Parameters

<i>base</i>	FlexIO I2S base pointer
-------------	-------------------------

Returns

Data read from data register.

**23.5.6.18 void FLEXIO_I2S_TransferTxCreateHandle (FLEXIO_I2S_Type * *base*,
flexio_i2s_handle_t * *handle*, flexio_i2s_callback_t *callback*, void * *userData*)**

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

<i>base</i>	Pointer to FLEXIO_I2S_Type structure
<i>handle</i>	Pointer to flexio_i2s_handle_t structure to store the transfer state.
<i>callback</i>	FlexIO I2S callback function, which is called while finished a block.
<i>userData</i>	User parameter for the FlexIO I2S callback.

23.5.6.19 void FLEXIO_I2S_TransferSetFormat (FLEXIO_I2S_Type * *base*, flexio_i2s_handle_t * *handle*, flexio_i2s_format_t * *format*, uint32_t *srcClock_Hz*)

Audio format can be changed at run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

<i>base</i>	Pointer to FLEXIO_I2S_Type structure.
<i>handle</i>	FlexIO I2S handle pointer.
<i>format</i>	Pointer to audio data format structure.
<i>srcClock_Hz</i>	FlexIO I2S bit clock source frequency in Hz. This parameter should be 0 while in slave mode.

23.5.6.20 void FLEXIO_I2S_TransferRxCreateHandle (FLEXIO_I2S_Type * *base*, flexio_i2s_handle_t * *handle*, flexio_i2s_callback_t *callback*, void * *userData*)

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

<i>base</i>	Pointer to FLEXIO_I2S_Type structure.
<i>handle</i>	Pointer to flexio_i2s_handle_t structure to store the transfer state.
<i>callback</i>	FlexIO I2S callback function, which is called while finished a block.
<i>userData</i>	User parameter for the FlexIO I2S callback.

23.5.6.21 status_t FLEXIO_I2S_TransferSendNonBlocking (FLEXIO_I2S_Type * *base*, flexio_i2s_handle_t * *handle*, flexio_i2s_transfer_t * *xfer*)

Note

The API returns immediately after transfer initiates. Call FLEXIO_I2S_GetRemainingBytes to poll the transfer status and check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

<i>base</i>	Pointer to FLEXIO_I2S_Type structure.
<i>handle</i>	Pointer to flexio_i2s_handle_t structure which stores the transfer state
<i>xfer</i>	Pointer to flexio_i2s_transfer_t structure

Return values

<i>kStatus_Success</i>	Successfully start the data transmission.
<i>kStatus_FLEXIO_I2S_Tx-Busy</i>	Previous transmission still not finished, data not all written to TX register yet.
<i>kStatus_InvalidArgument</i>	The input parameter is invalid.

23.5.6.22 status_t FLEXIO_I2S_TransferReceiveNonBlocking (FLEXIO_I2S_Type * base, flexio_i2s_handle_t * handle, flexio_i2s_transfer_t * xfer)

Note

The API returns immediately after transfer initiates. Call FLEXIO_I2S_GetRemainingBytes to poll the transfer status to check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

<i>base</i>	Pointer to FLEXIO_I2S_Type structure.
<i>handle</i>	Pointer to flexio_i2s_handle_t structure which stores the transfer state
<i>xfer</i>	Pointer to flexio_i2s_transfer_t structure

Return values

<i>kStatus_Success</i>	Successfully start the data receive.
------------------------	--------------------------------------

<i>kStatus_FLEXIO_I2S_-RxBusy</i>	Previous receive still not finished.
<i>kStatus_InvalidArgument</i>	The input parameter is invalid.

23.5.6.23 void FLEXIO_I2S_TransferAbortSend (FLEXIO_I2S_Type * *base*, flexio_i2s_handle_t * *handle*)

Note

This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

<i>base</i>	Pointer to FLEXIO_I2S_Type structure.
<i>handle</i>	Pointer to flexio_i2s_handle_t structure which stores the transfer state

23.5.6.24 void FLEXIO_I2S_TransferAbortReceive (FLEXIO_I2S_Type * *base*, flexio_i2s_handle_t * *handle*)

Note

This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

<i>base</i>	Pointer to FLEXIO_I2S_Type structure.
<i>handle</i>	Pointer to flexio_i2s_handle_t structure which stores the transfer state

23.5.6.25 status_t FLEXIO_I2S_TransferGetSendCount (FLEXIO_I2S_Type * *base*, flexio_i2s_handle_t * *handle*, size_t * *count*)

Parameters

<i>base</i>	Pointer to FLEXIO_I2S_Type structure.
<i>handle</i>	Pointer to flexio_i2s_handle_t structure which stores the transfer state
<i>count</i>	Bytes sent.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is not a non-blocking transaction currently in progress.

23.5.6.26 `status_t FLEXIO_I2S_TransferGetReceiveCount (FLEXIO_I2S_Type * base, flexio_i2s_handle_t * handle, size_t * count)`

Parameters

<i>base</i>	Pointer to FLEXIO_I2S_Type structure.
<i>handle</i>	Pointer to flexio_i2s_handle_t structure which stores the transfer state
<i>count</i>	Bytes recieved.

Returns

count Bytes received.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is not a non-blocking transaction currently in progress.

23.5.6.27 `void FLEXIO_I2S_TransferTxHandleIRQ (void * i2sBase, void * i2sHandle)`

Parameters

<i>i2sBase</i>	Pointer to FLEXIO_I2S_Type structure.
<i>i2sHandle</i>	Pointer to flexio_i2s_handle_t structure

23.5.6.28 `void FLEXIO_I2S_TransferRxHandleIRQ (void * i2sBase, void * i2sHandle)`

Parameters

<i>i2sBase</i>	Pointer to FLEXIO_I2S_Type structure.
<i>i2sHandle</i>	Pointer to flexio_i2s_handle_t structure.

23.5.7 FlexIO eDMA I2S Driver

23.5.7.1 Overview

Data Structures

- struct `flexio_i2s_edma_handle_t`
FlexIO I2S DMA transfer handle, users should not touch the content of the handle. [More...](#)

Typedefs

- typedef void(* `flexio_i2s_edma_callback_t`)(`FLEXIO_I2S_Type` *base, `flexio_i2s_edma_handle_t` *handle, `status_t` status, void *userData)
FlexIO I2S eDMA transfer callback function for finish and error.

Driver version

- #define `FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 7)`)
FlexIO I2S EDMA driver version 2.1.7.

eDMA Transactional

- void `FLEXIO_I2S_TransferTxCreateHandleEDMA` (`FLEXIO_I2S_Type` *base, `flexio_i2s_edma_handle_t` *handle, `flexio_i2s_edma_callback_t` callback, void *userData, `edma_handle_t` *dmaHandle)
Initializes the FlexIO I2S eDMA handle.
- void `FLEXIO_I2S_TransferRxCreateHandleEDMA` (`FLEXIO_I2S_Type` *base, `flexio_i2s_edma_handle_t` *handle, `flexio_i2s_edma_callback_t` callback, void *userData, `edma_handle_t` *dmaHandle)
Initializes the FlexIO I2S Rx eDMA handle.
- void `FLEXIO_I2S_TransferSetFormatEDMA` (`FLEXIO_I2S_Type` *base, `flexio_i2s_edma_handle_t` *handle, `flexio_i2s_format_t` *format, `uint32_t` srcClock_Hz)
Configures the FlexIO I2S Tx audio format.
- `status_t` `FLEXIO_I2S_TransferSendEDMA` (`FLEXIO_I2S_Type` *base, `flexio_i2s_edma_handle_t` *handle, `flexio_i2s_transfer_t` *xfer)
Performs a non-blocking FlexIO I2S transfer using DMA.
- `status_t` `FLEXIO_I2S_TransferReceiveEDMA` (`FLEXIO_I2S_Type` *base, `flexio_i2s_edma_handle_t` *handle, `flexio_i2s_transfer_t` *xfer)
Performs a non-blocking FlexIO I2S receive using eDMA.
- void `FLEXIO_I2S_TransferAbortSendEDMA` (`FLEXIO_I2S_Type` *base, `flexio_i2s_edma_handle_t` *handle)
Aborts a FlexIO I2S transfer using eDMA.
- void `FLEXIO_I2S_TransferAbortReceiveEDMA` (`FLEXIO_I2S_Type` *base, `flexio_i2s_edma_handle_t` *handle)
Aborts a FlexIO I2S receive using eDMA.

- `status_t FLEXIO_I2S_TransferGetSendCountEDMA` (`FLEXIO_I2S_Type *base`, `flexio_i2s_edma_handle_t *handle`, `size_t *count`)
Gets the remaining bytes to be sent.
- `status_t FLEXIO_I2S_TransferGetReceiveCountEDMA` (`FLEXIO_I2S_Type *base`, `flexio_i2s_edma_handle_t *handle`, `size_t *count`)
Get the remaining bytes to be received.

23.5.7.2 Data Structure Documentation

23.5.7.2.1 struct flexio_i2s_edma_handle

Data Fields

- `edma_handle_t * dmaHandle`
DMA handler for FlexIO I2S send.
- `uint8_t bytesPerFrame`
Bytes in a frame.
- `uint8_t nbytes`
eDMA minor byte transfer count initially configured.
- `uint32_t state`
Internal state for FlexIO I2S eDMA transfer.
- `flexio_i2s_edma_callback_t callback`
Callback for users while transfer finish or error occurred.
- `void * userData`
User callback parameter.
- `edma_tcd_t tcd [FLEXIO_I2S_XFER_QUEUE_SIZE+1U]`
TCD pool for eDMA transfer.
- `flexio_i2s_transfer_t queue [FLEXIO_I2S_XFER_QUEUE_SIZE]`
Transfer queue storing queued transfer.
- `size_t transferSize [FLEXIO_I2S_XFER_QUEUE_SIZE]`
Data bytes need to transfer.
- `volatile uint8_t queueUser`
Index for user to queue transfer.
- `volatile uint8_t queueDriver`
Index for driver to get the transfer data and size.

Field Documentation

- (1) `uint8_t flexio_i2s_edma_handle_t::nbytes`
- (2) `edma_tcd_t flexio_i2s_edma_handle_t::tcd[FLEXIO_I2S_XFER_QUEUE_SIZE+1U]`
- (3) `flexio_i2s_transfer_t flexio_i2s_edma_handle_t::queue[FLEXIO_I2S_XFER_QUEUE_SIZE]`
- (4) `volatile uint8_t flexio_i2s_edma_handle_t::queueUser`

23.5.7.3 Macro Definition Documentation

23.5.7.3.1 `#define FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 7))`

23.5.7.4 Function Documentation

23.5.7.4.1 `void FLEXIO_I2S_TransferTxCreateHandleEDMA (FLEXIO_I2S_Type * base, flexio_i2s_edma_handle_t * handle, flexio_i2s_edma_callback_t callback, void * userData, edma_handle_t * dmaHandle)`

This function initializes the FlexIO I2S master DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

<i>base</i>	FlexIO I2S peripheral base address.
<i>handle</i>	FlexIO I2S eDMA handle pointer.
<i>callback</i>	FlexIO I2S eDMA callback function called while finished a block.
<i>userData</i>	User parameter for callback.
<i>dmaHandle</i>	eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

23.5.7.4.2 `void FLEXIO_I2S_TransferRxCreateHandleEDMA (FLEXIO_I2S_Type * base, flexio_i2s_edma_handle_t * handle, flexio_i2s_edma_callback_t callback, void * userData, edma_handle_t * dmaHandle)`

This function initializes the FlexIO I2S slave DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

<i>base</i>	FlexIO I2S peripheral base address.
<i>handle</i>	FlexIO I2S eDMA handle pointer.
<i>callback</i>	FlexIO I2S eDMA callback function called while finished a block.
<i>userData</i>	User parameter for callback.
<i>dmaHandle</i>	eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

23.5.7.4.3 void FLEXIO_I2S_TransferSetFormatEDMA (FLEXIO_I2S_Type * *base*, flexio_i2s_edma_handle_t * *handle*, flexio_i2s_format_t * *format*, uint32_t *srcClock_Hz*)

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to format.

Parameters

<i>base</i>	FlexIO I2S peripheral base address.
<i>handle</i>	FlexIO I2S eDMA handle pointer
<i>format</i>	Pointer to FlexIO I2S audio data format structure.
<i>srcClock_Hz</i>	FlexIO I2S clock source frequency in Hz, it should be 0 while in slave mode.

23.5.7.4.4 status_t FLEXIO_I2S_TransferSendEDMA (FLEXIO_I2S_Type * *base*, flexio_i2s_edma_handle_t * *handle*, flexio_i2s_transfer_t * *xfer*)

Note

This interface returned immediately after transfer initiates. Users should call FLEXIO_I2S_GetTransferStatus to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

<i>base</i>	FlexIO I2S peripheral base address.
<i>handle</i>	FlexIO I2S DMA handle pointer.
<i>xfer</i>	Pointer to DMA transfer structure.

Return values

<i>kStatus_Success</i>	Start a FlexIO I2S eDMA send successfully.
<i>kStatus_InvalidArgument</i>	The input arguments is invalid.
<i>kStatus_TxBusy</i>	FlexIO I2S is busy sending data.

23.5.7.4.5 **status_t FLEXIO_I2S_TransferReceiveEDMA (FLEXIO_I2S_Type * *base*, flexio_i2s_edma_handle_t * *handle*, flexio_i2s_transfer_t * *xfer*)**

Note

This interface returned immediately after transfer initiates. Users should call FLEXIO_I2S_GetReceiveRemainingBytes to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

<i>base</i>	FlexIO I2S peripheral base address.
<i>handle</i>	FlexIO I2S DMA handle pointer.
<i>xfer</i>	Pointer to DMA transfer structure.

Return values

<i>kStatus_Success</i>	Start a FlexIO I2S eDMA receive successfully.
<i>kStatus_InvalidArgument</i>	The input arguments is invalid.
<i>kStatus_RxBusy</i>	FlexIO I2S is busy receiving data.

23.5.7.4.6 **void FLEXIO_I2S_TransferAbortSendEDMA (FLEXIO_I2S_Type * *base*, flexio_i2s_edma_handle_t * *handle*)**

Parameters

<i>base</i>	FlexIO I2S peripheral base address.
<i>handle</i>	FlexIO I2S DMA handle pointer.

23.5.7.4.7 **void FLEXIO_I2S_TransferAbortReceiveEDMA (FLEXIO_I2S_Type * *base*, flexio_i2s_edma_handle_t * *handle*)**

Parameters

<i>base</i>	FlexIO I2S peripheral base address.
<i>handle</i>	FlexIO I2S DMA handle pointer.

23.5.7.4.8 `status_t FLEXIO_I2S_TransferGetSendCountEDMA (FLEXIO_I2S_Type * base, flexio_i2s_edma_handle_t * handle, size_t * count)`

Parameters

<i>base</i>	FlexIO I2S peripheral base address.
<i>handle</i>	FlexIO I2S DMA handle pointer.
<i>count</i>	Bytes sent.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is not a non-blocking transaction currently in progress.

23.5.7.4.9 `status_t FLEXIO_I2S_TransferGetReceiveCountEDMA (FLEXIO_I2S_Type * base, flexio_i2s_edma_handle_t * handle, size_t * count)`

Parameters

<i>base</i>	FlexIO I2S peripheral base address.
<i>handle</i>	FlexIO I2S DMA handle pointer.
<i>count</i>	Bytes received.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is not a non-blocking transaction currently in progress.

23.6 FlexIO MCU Interface LCD Driver

23.6.1 Overview

The MCUXpresso SDK provides a peripheral driver for LCD (8080 or 6800 interface) function using Flexible I/O module of MCUXpresso SDK devices.

The FlexIO LCD driver supports both 8-bit and 16-bit data bus, 8080 and 6800 interface. User could change the macro `FLEXIO_MCULCD_DATA_BUS_WIDTH` to choose 8-bit data bus or 16-bit data bus.

The FlexIO LCD driver supports three kinds of data transfer:

1. Send a data array. For example, send the LCD image data to the LCD controller.
2. Send a value many times. For example, send 0 many times to clean the LCD screen.
3. Read data into a data array. For example, read image from LCD controller.

The FlexIO LCD driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for FlexIO LCD initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FlexIO LCD peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. FlexIO LCD functional operation groups provide the functional APIs set.

Transactional APIs are transaction target high level APIs. The transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs can satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code.

Transactional APIs support asynchronous transfer. This means that the function `FLEXIO_MCULCD_-TransferNonBlocking` sets up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_FLEXIO_MCULCD_Idle` status.

23.6.2 Typical use case

23.6.2.1 FlexIO LCD send/receive using functional APIs

This example shows how to send command, or write and read data using the functional APIs. The data bus is 16-bit.

```
uint16_t dataToSend[] = { ... };
uint16_t dataToReceive[] = { ... };

FLEXIO_MCULCD_Type flexioLcdDev;
flexio_MCULCD_transfer_t xfer;
flexio_MCULCD_config_t config;

FLEXIO_MCULCD_GetDefaultConfig(&config);
FLEXIO_MCULCD_Init(&flexioLcdDev, &config, 12000000);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
```

```

FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command1);
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command1;
xfer.dataCount = 0; // Only send command, no data transfer.
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command2);
FLEXIO_MCULCD_WriteDataArrayBlocking(&flexioLcdDev, dataToSend, sizeof(
    dataToSend));
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command2;
xfer.mode = kFLEXIO_MCULCD_WriteArray;
xfer.dataAddrOrSameValue = (uint32_t)dataToSend;
xfer.dataCount = sizeof(dataToSend);
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command2);
FLEXIO_MCULCD_WriteSameValueBlocking(&flexioLcdDev, value, 1000); //
    Send value 1000 times
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command2;
xfer.mode = kFLEXIO_MCULCD_WriteSameValue;
xfer.dataAddrOrSameValue = value;
xfer.dataCount = 1000;
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command3);
FLEXIO_MCULCD_ReadDataArrayBlocking(&flexioLcdDev, dataToReceive, sizeof(
    dataToReceive));
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command3;
xfer.mode = kFLEXIO_MCULCD_ReadArray;
xfer.dataAddrOrSameValue = (uint32_t)dataToReceive;
xfer.dataCount = sizeof(dataToReceive);
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);

```

23.6.2.2 FlexIO LCD send/receive using interrupt transactional APIs

```

flexio_MCULCD_handle_t handle;
volatile bool completeFlag = false;

void flexioLcdCallback(FLEXIO_MCULCD_Type *base, flexio_MCULCD_handle_t *handle,
    status_t status, void *userData)
{
    if (kStatus_FLEXIO_MCULCD_Idle == status)
    {
        completeFlag = true;
    }
}

void main(void)

```

```

{
    // Init the FlexIO LCD driver.
    FLEXIO_MCULCD_Init(...);

    // Create the transactional handle.
    FLEXIO_MCULCD_TransferCreateHandle(&flexioLcdDev, &handle,
        flexioLcdCallback, NULL);

    xfer.command = command1;
    xfer.dataCount = 0; // Only send command, no data transfer.
    completeFlag = false;
    FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &xfer);

    // When only send method, it is not necessary to wait for the callback,
    // because the command is sent using a blocking method internally. The
    // command has been sent out after the function FLEXIO_MCULCD_TransferNonBlocking
    // returns.
    while (!completeFlag)
    {
    }

    xfer.command = command2;
    xfer.mode = kFLEXIO_MCULCD_WriteArray;
    xfer.dataAddrOrSameValue = (uint32_t)dataToSend;
    xfer.dataCount = sizeof(dataToSend);
    completeFlag = false;
    FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &handle, &xfer);

    while (!completeFlag)
    {
    }

    xfer.command = command2;
    xfer.mode = kFLEXIO_MCULCD_WriteSameValue;
    xfer.dataAddrOrSameValue = value;
    xfer.dataCount = 1000;
    completeFlag = false;
    FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &handle, &xfer);

    while (!completeFlag)
    {
    }

    xfer.command = command3;
    xfer.mode = kFLEXIO_MCULCD_ReadArray;
    xfer.dataAddrOrSameValue = (uint32_t)dataToReceive;
    xfer.dataCount = sizeof(dataToReceive);
    completeFlag = false;
    FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &handle, &xfer);

    while (!completeFlag)
    {
    }
}

```

Modules

- [FlexIO eDMA MCU Interface LCD Driver](#)

SDK provide eDMA transactional APIs to transfer data using eDMA, the eDMA method is similar with interrupt transactional method.

Data Structures

- struct [FLEXIO_MCULCD_Type](#)
Define FlexIO MCULCD access structure typedef. [More...](#)
- struct [flexio_mculcd_config_t](#)
Define FlexIO MCULCD configuration structure. [More...](#)
- struct [flexio_mculcd_transfer_t](#)
Define FlexIO MCULCD transfer structure. [More...](#)
- struct [flexio_mculcd_handle_t](#)
Define FlexIO MCULCD handle structure. [More...](#)

Macros

- #define [FLEXIO_MCULCD_WAIT_COMPLETE_TIME](#) 512
The delay time to wait for FLEXIO transmit complete.
- #define [FLEXIO_MCULCD_DATA_BUS_WIDTH](#) 16UL
The data bus width, must be 8 or 16.

Typedefs

- typedef void(* [flexio_mculcd_pin_func_t](#))(bool set)
Function to set or clear the CS and RS pin.
- typedef void(* [flexio_mculcd_transfer_callback_t](#))([FLEXIO_MCULCD_Type](#) *base, [flexio_mculcd_handle_t](#) *handle, [status_t](#) status, void *userData)
FlexIO MCULCD callback for finished transfer.

Enumerations

- enum {
 [kStatus_FLEXIO_MCULCD_Idle](#) = MAKE_STATUS(kStatusGroup_FLEXIO_MCULCD, 0),
 [kStatus_FLEXIO_MCULCD_Busy](#) = MAKE_STATUS(kStatusGroup_FLEXIO_MCULCD, 1),
 [kStatus_FLEXIO_MCULCD_Error](#) = MAKE_STATUS(kStatusGroup_FLEXIO_MCULCD, 2) }
FlexIO LCD transfer status.
- enum [flexio_mculcd_pixel_format_t](#) {
 [kFLEXIO_MCULCD_RGB565](#) = 0,
 [kFLEXIO_MCULCD_BGR565](#),
 [kFLEXIO_MCULCD_RGB888](#),
 [kFLEXIO_MCULCD_BGR888](#) }
Define FlexIO MCULCD pixel format.
- enum [flexio_mculcd_bus_t](#) {
 [kFLEXIO_MCULCD_8080](#),
 [kFLEXIO_MCULCD_6800](#) }
Define FlexIO MCULCD bus type.
- enum [_flexio_mculcd_interrupt_enable](#) {
 [kFLEXIO_MCULCD_TxEmptyInterruptEnable](#) = (1U << 0U),

- ```
kFLEXIO_MCULCD_RxFullInterruptEnable = (1U << 1U) }
```
- *Define FlexIO MCULCD interrupt mask.*
  - enum `_flexio_mculcd_status_flags` {
 

```
kFLEXIO_MCULCD_TxEmptyFlag = (1U << 0U),
kFLEXIO_MCULCD_RxFullFlag = (1U << 1U) }
```
  - *Define FlexIO MCULCD status mask.*
  - enum `_flexio_mculcd_dma_enable` {
 

```
kFLEXIO_MCULCD_TxDmaEnable = 0x1U,
kFLEXIO_MCULCD_RxDmaEnable = 0x2U }
```
  - *Define FlexIO MCULCD DMA mask.*
  - enum `flexio_mculcd_transfer_mode_t` {
 

```
kFLEXIO_MCULCD_ReadArray,
kFLEXIO_MCULCD_WriteArray,
kFLEXIO_MCULCD_WriteSameValue }
```
  - *Transfer mode.*

## Driver version

- #define `FSL_FLEXIO_MCULCD_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 8)`)  
*FlexIO MCULCD driver version.*

## FlexIO MCULCD Configuration

- `status_t` `FLEXIO_MCULCD_Init` (`FLEXIO_MCULCD_Type` \*base, `flexio_mculcd_config_t` \*config, `uint32_t` srcClock\_Hz)  
*Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO MCULCD hardware, and configures the FlexIO MCULCD with FlexIO MCULCD configuration.*
- void `FLEXIO_MCULCD_Deinit` (`FLEXIO_MCULCD_Type` \*base)  
*Resets the FLEXIO\_MCULCD timer and shifter configuration.*
- void `FLEXIO_MCULCD_GetDefaultConfig` (`flexio_mculcd_config_t` \*config)  
*Gets the default configuration to configure the FlexIO MCULCD.*

## Status

- `uint32_t` `FLEXIO_MCULCD_GetStatusFlags` (`FLEXIO_MCULCD_Type` \*base)  
*Gets FlexIO MCULCD status flags.*
- void `FLEXIO_MCULCD_ClearStatusFlags` (`FLEXIO_MCULCD_Type` \*base, `uint32_t` mask)  
*Clears FlexIO MCULCD status flags.*

## Interrupts

- void `FLEXIO_MCULCD_EnableInterrupts` (`FLEXIO_MCULCD_Type` \*base, `uint32_t` mask)  
*Enables the FlexIO MCULCD interrupt.*
- void `FLEXIO_MCULCD_DisableInterrupts` (`FLEXIO_MCULCD_Type` \*base, `uint32_t` mask)

*Disables the FlexIO MCULCD interrupt.*

## DMA Control

- static void `FLEXIO_MCULCD_EnableTxDMA` (`FLEXIO_MCULCD_Type *base`, bool enable)  
*Enables/disables the FlexIO MCULCD transmit DMA.*
- static void `FLEXIO_MCULCD_EnableRxDMA` (`FLEXIO_MCULCD_Type *base`, bool enable)  
*Enables/disables the FlexIO MCULCD receive DMA.*
- static uint32\_t `FLEXIO_MCULCD_GetTxDataRegisterAddress` (`FLEXIO_MCULCD_Type *base`)  
*Gets the FlexIO MCULCD transmit data register address.*
- static uint32\_t `FLEXIO_MCULCD_GetRxDataRegisterAddress` (`FLEXIO_MCULCD_Type *base`)  
*Gets the FlexIO MCULCD receive data register address.*

## Bus Operations

- `status_t FLEXIO_MCULCD_SetBaudRate` (`FLEXIO_MCULCD_Type *base`, uint32\_t baudRate-\_Bps, uint32\_t srcClock\_Hz)  
*Set desired baud rate.*
- void `FLEXIO_MCULCD_SetSingleBeatWriteConfig` (`FLEXIO_MCULCD_Type *base`)  
*Configures the FLEXIO MCULCD to multiple beats write mode.*
- void `FLEXIO_MCULCD_ClearSingleBeatWriteConfig` (`FLEXIO_MCULCD_Type *base`)  
*Clear the FLEXIO MCULCD multiple beats write mode configuration.*
- void `FLEXIO_MCULCD_SetSingleBeatReadConfig` (`FLEXIO_MCULCD_Type *base`)  
*Configures the FLEXIO MCULCD to multiple beats read mode.*
- void `FLEXIO_MCULCD_ClearSingleBeatReadConfig` (`FLEXIO_MCULCD_Type *base`)  
*Clear the FLEXIO MCULCD multiple beats read mode configuration.*
- void `FLEXIO_MCULCD_SetMultiBeatsWriteConfig` (`FLEXIO_MCULCD_Type *base`)  
*Configures the FLEXIO MCULCD to multiple beats write mode.*
- void `FLEXIO_MCULCD_ClearMultiBeatsWriteConfig` (`FLEXIO_MCULCD_Type *base`)  
*Clear the FLEXIO MCULCD multiple beats write mode configuration.*
- void `FLEXIO_MCULCD_SetMultiBeatsReadConfig` (`FLEXIO_MCULCD_Type *base`)  
*Configures the FLEXIO MCULCD to multiple beats read mode.*
- void `FLEXIO_MCULCD_ClearMultiBeatsReadConfig` (`FLEXIO_MCULCD_Type *base`)  
*Clear the FLEXIO MCULCD multiple beats read mode configuration.*
- static void `FLEXIO_MCULCD_Enable` (`FLEXIO_MCULCD_Type *base`, bool enable)  
*Enables/disables the FlexIO MCULCD module operation.*
- uint32\_t `FLEXIO_MCULCD_ReadData` (`FLEXIO_MCULCD_Type *base`)  
*Read data from the FLEXIO MCULCD RX shifter buffer.*
- static void `FLEXIO_MCULCD_WriteData` (`FLEXIO_MCULCD_Type *base`, uint32\_t data)  
*Write data into the FLEXIO MCULCD TX shifter buffer.*
- static void `FLEXIO_MCULCD_StartTransfer` (`FLEXIO_MCULCD_Type *base`)  
*Assert the nCS to start transfer.*
- static void `FLEXIO_MCULCD_StopTransfer` (`FLEXIO_MCULCD_Type *base`)  
*De-assert the nCS to stop transfer.*
- void `FLEXIO_MCULCD_WaitTransmitComplete` (void)  
*Wait for transmit data send out finished.*

- void `FLEXIO_MCULCD_WriteCommandBlocking` (`FLEXIO_MCULCD_Type *base`, `uint32_t command`)  
*Send command in blocking way.*
- void `FLEXIO_MCULCD_WriteDataArrayBlocking` (`FLEXIO_MCULCD_Type *base`, `const void *data`, `size_t size`)  
*Send data array in blocking way.*
- void `FLEXIO_MCULCD_ReadDataArrayBlocking` (`FLEXIO_MCULCD_Type *base`, `void *data`, `size_t size`)  
*Read data into array in blocking way.*
- void `FLEXIO_MCULCD_WriteSameValueBlocking` (`FLEXIO_MCULCD_Type *base`, `uint32_t sameValue`, `size_t size`)  
*Send the same value many times in blocking way.*
- void `FLEXIO_MCULCD_TransferBlocking` (`FLEXIO_MCULCD_Type *base`, `flexio_mculcd_transfer_t *xfer`)  
*Performs a polling transfer.*

## Transactional

- `status_t FLEXIO_MCULCD_TransferCreateHandle` (`FLEXIO_MCULCD_Type *base`, `flexio_mculcd_handle_t *handle`, `flexio_mculcd_transfer_callback_t callback`, `void *userData`)  
*Initializes the FlexIO MCULCD handle, which is used in transactional functions.*
- `status_t FLEXIO_MCULCD_TransferNonBlocking` (`FLEXIO_MCULCD_Type *base`, `flexio_mculcd_handle_t *handle`, `flexio_mculcd_transfer_t *xfer`)  
*Transfer data using IRQ.*
- void `FLEXIO_MCULCD_TransferAbort` (`FLEXIO_MCULCD_Type *base`, `flexio_mculcd_handle_t *handle`)  
*Aborts the data transfer, which used IRQ.*
- `status_t FLEXIO_MCULCD_TransferGetCount` (`FLEXIO_MCULCD_Type *base`, `flexio_mculcd_handle_t *handle`, `size_t *count`)  
*Gets the data transfer status which used IRQ.*
- void `FLEXIO_MCULCD_TransferHandleIRQ` (`void *base`, `void *handle`)  
*FlexIO MCULCD IRQ handler function.*

## 23.6.3 Data Structure Documentation

### 23.6.3.1 struct FLEXIO\_MCULCD\_Type

#### Data Fields

- `FLEXIO_Type * flexioBase`  
*FlexIO base pointer.*
- `flexio_mculcd_bus_t busType`  
*The bus type, 8080 or 6800.*
- `uint8_t dataPinStartIndex`  
*Start index of the data pin, the FlexIO pin `dataPinStartIndex` to `(dataPinStartIndex + FLEXIO_MCULCD_DATA_BUS_WIDTH - 1)` will be used for data transfer.*



- `uint8_t ENWRPinIndex`  
*Pin select for WR(8080 mode), EN(6800 mode).*
- `uint8_t RDPinIndex`  
*Pin select for RD(8080 mode), not used in 6800 mode.*
- `uint8_t txShifterStartIndex`  
*Start index of shifters used for data write, it must be 0 or 4.*
- `uint8_t txShifterEndIndex`  
*End index of shifters used for data write.*
- `uint8_t rxShifterStartIndex`  
*Start index of shifters used for data read.*
- `uint8_t rxShifterEndIndex`  
*End index of shifters used for data read, it must be 3 or 7.*
- `uint8_t timerIndex`  
*Timer index used in FlexIO MCULCD.*
- `flexio_mculcd_pin_func_t setCSPin`  
*Function to set or clear the CS pin.*
- `flexio_mculcd_pin_func_t setRSPin`  
*Function to set or clear the RS pin.*
- `flexio_mculcd_pin_func_t setRDWRPin`  
*Function to set or clear the RD/WR pin, only used in 6800 mode.*

### Field Documentation

- (1) `FLEXIO_Type* FLEXIO_MCULCD_Type::flexioBase`
- (2) `flexio_mculcd_bus_t FLEXIO_MCULCD_Type::busType`
- (3) `uint8_t FLEXIO_MCULCD_Type::dataPinStartIndex`

Only support data bus width 8 and 16.

- (4) `uint8_t FLEXIO_MCULCD_Type::ENWRPinIndex`
- (5) `uint8_t FLEXIO_MCULCD_Type::RDPinIndex`
- (6) `uint8_t FLEXIO_MCULCD_Type::txShifterStartIndex`
- (7) `uint8_t FLEXIO_MCULCD_Type::txShifterEndIndex`
- (8) `uint8_t FLEXIO_MCULCD_Type::rxShifterStartIndex`
- (9) `uint8_t FLEXIO_MCULCD_Type::rxShifterEndIndex`
- (10) `uint8_t FLEXIO_MCULCD_Type::timerIndex`
- (11) `flexio_mculcd_pin_func_t FLEXIO_MCULCD_Type::setCSPin`
- (12) `flexio_mculcd_pin_func_t FLEXIO_MCULCD_Type::setRSPin`
- (13) `flexio_mculcd_pin_func_t FLEXIO_MCULCD_Type::setRDWRPin`

### 23.6.3.2 struct `flexio_mculcd_config_t`

#### Data Fields

- `bool enable`  
*Enable/disable FlexIO MCULCD after configuration.*
- `bool enableInDoze`  
*Enable/disable FlexIO operation in doze mode.*
- `bool enableInDebug`  
*Enable/disable FlexIO operation in debug mode.*
- `bool enableFastAccess`  
*Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- `uint32_t baudRate_Bps`  
*Baud rate in bit-per-second for all data lines combined.*

**Field Documentation**

- (1) `bool flexio_mculcd_config_t::enable`
- (2) `bool flexio_mculcd_config_t::enableInDoze`
- (3) `bool flexio_mculcd_config_t::enableInDebug`
- (4) `bool flexio_mculcd_config_t::enableFastAccess`
- (5) `uint32_t flexio_mculcd_config_t::baudRate_Bps`

**23.6.3.3 struct flexio\_mculcd\_transfer\_t****Data Fields**

- `uint32_t command`  
*Command to send.*
- `flexio_mculcd_transfer_mode_t mode`  
*Transfer mode.*
- `uint32_t dataAddrOrSameValue`  
*When sending the same value for many times, this is the value to send.*
- `size_t dataSize`  
*How many bytes to transfer.*

**Field Documentation**

- (1) `uint32_t flexio_mculcd_transfer_t::command`
- (2) `flexio_mculcd_transfer_mode_t flexio_mculcd_transfer_t::mode`
- (3) `uint32_t flexio_mculcd_transfer_t::dataAddrOrSameValue`

When writing or reading array, this is the address of the data array.

- (4) `size_t flexio_mculcd_transfer_t::dataSize`

**23.6.3.4 struct \_flexio\_mculcd\_handle**

typedef for `flexio_mculcd_handle_t` in advance.

**Data Fields**

- `uint32_t dataAddrOrSameValue`  
*When sending the same value for many times, this is the value to send.*
- `size_t dataCount`  
*Total count to be transferred.*
- `volatile size_t remainingCount`  
*Remaining count to transfer.*
- `volatile uint32_t state`

- *FlexIO MCULCD internal state.*  
flexio\_mculcd\_transfer\_callback\_t completionCallback  
*FlexIO MCULCD transfer completed callback.*
- void \* userData  
*Callback parameter.*

### Field Documentation

#### (1) uint32\_t flexio\_mculcd\_handle\_t::dataAddrOrSameValue

When writing or reading array, this is the address of the data array.

#### (2) size\_t flexio\_mculcd\_handle\_t::dataCount

#### (3) volatile size\_t flexio\_mculcd\_handle\_t::remainingCount

#### (4) volatile uint32\_t flexio\_mculcd\_handle\_t::state

#### (5) flexio\_mculcd\_transfer\_callback\_t flexio\_mculcd\_handle\_t::completionCallback

#### (6) void\* flexio\_mculcd\_handle\_t::userData

### 23.6.4 Macro Definition Documentation

#### 23.6.4.1 #define FSL\_FLEXIO\_MCULCD\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 8))

#### 23.6.4.2 #define FLEXIO\_MCULCD\_WAIT\_COMPLETE\_TIME 512

Currently there is no method to detect whether the data has been sent out from the shifter, so the driver use a software delay for this. When the data is written to shifter buffer, the driver call the delay function to wait for the data shift out. If this value is too small, then the last few bytes might be lost when writing data using interrupt method or DMA method.

### 23.6.5 Typedef Documentation

#### 23.6.5.1 typedef void(\* flexio\_mculcd\_pin\_func\_t)(bool set)

#### 23.6.5.2 typedef void(\* flexio\_mculcd\_transfer\_callback\_t)(FLEXIO\_MCULCD\_Type \*base, flexio\_mculcd\_handle\_t \*handle, status\_t status, void \*userData)

When transfer finished, the callback function is called and returns the `status` as `kStatus_FLEXIO_MCULCD_Idle`.

## 23.6.6 Enumeration Type Documentation

### 23.6.6.1 anonymous enum

Enumerator

*kStatus\_FLEXIO\_MCULCD\_Idle* FlexIO LCD is idle.  
*kStatus\_FLEXIO\_MCULCD\_Busy* FlexIO LCD is busy.  
*kStatus\_FLEXIO\_MCULCD\_Error* FlexIO LCD error occurred.

### 23.6.6.2 enum flexio\_mculcd\_pixel\_format\_t

Enumerator

*kFLEXIO\_MCULCD\_RGB565* RGB565, 16-bit.  
*kFLEXIO\_MCULCD\_BGR565* BGR565, 16-bit.  
*kFLEXIO\_MCULCD\_RGB888* RGB888, 24-bit.  
*kFLEXIO\_MCULCD\_BGR888* BGR888, 24-bit.

### 23.6.6.3 enum flexio\_mculcd\_bus\_t

Enumerator

*kFLEXIO\_MCULCD\_8080* Using Intel 8080 bus.  
*kFLEXIO\_MCULCD\_6800* Using Motorola 6800 bus.

### 23.6.6.4 enum flexio\_mculcd\_interrupt\_enable

Enumerator

*kFLEXIO\_MCULCD\_TxEmptyInterruptEnable* Transmit buffer empty interrupt enable.  
*kFLEXIO\_MCULCD\_RxFullInterruptEnable* Receive buffer full interrupt enable.

### 23.6.6.5 enum flexio\_mculcd\_status\_flags

Enumerator

*kFLEXIO\_MCULCD\_TxEmptyFlag* Transmit buffer empty flag.  
*kFLEXIO\_MCULCD\_RxFullFlag* Receive buffer full flag.

### 23.6.6.6 enum flexio\_mculcd\_dma\_enable

Enumerator

*kFLEXIO\_MCULCD\_TxDmaEnable* Tx DMA request source.  
*kFLEXIO\_MCULCD\_RxDmaEnable* Rx DMA request source.

### 23.6.6.7 enum flexio\_mculcd\_transfer\_mode\_t

Enumerator

*kFLEXIO\_MCULCD\_ReadArray* Read data into an array.  
*kFLEXIO\_MCULCD\_WriteArray* Write data from an array.  
*kFLEXIO\_MCULCD\_WriteSameValue* Write the same value many times.

## 23.6.7 Function Documentation

### 23.6.7.1 status\_t FLEXIO\_MCULCD\_Init ( FLEXIO\_MCULCD\_Type \* *base*, flexio\_mculcd\_config\_t \* *config*, uint32\_t *srcClock\_Hz* )

The configuration structure can be filled by the user, or be set with default values by the [FLEXIO\\_MCULCD\\_GetDefaultConfig](#).

Parameters

|                    |                                                                  |
|--------------------|------------------------------------------------------------------|
| <i>base</i>        | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.     |
| <i>config</i>      | Pointer to the <a href="#">flexio_mculcd_config_t</a> structure. |
| <i>srcClock_Hz</i> | FlexIO source clock in Hz.                                       |

Return values

|                                |                                                    |
|--------------------------------|----------------------------------------------------|
| <i>kStatus_Success</i>         | Initialization success.                            |
| <i>kStatus_InvalidArgument</i> | Initialization failed because of invalid argument. |

### 23.6.7.2 void FLEXIO\_MCULCD\_Deinit ( FLEXIO\_MCULCD\_Type \* *base* )

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

### 23.6.7.3 void FLEXIO\_MCULCD\_GetDefaultConfig ( flexio\_mculcd\_config\_t \* config )

The default configuration value is:

```
* config->enable = true;
* config->enableInDoze = false;
* config->enableInDebug = true;
* config->enableFastAccess = true;
* config->baudRate_Bps = 96000000U;
*
```

Parameters

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| <i>config</i> | Pointer to the <a href="#">flexio_mculcd_config_t</a> structure. |
|---------------|------------------------------------------------------------------|

### 23.6.7.4 uint32\_t FLEXIO\_MCULCD\_GetStatusFlags ( FLEXIO\_MCULCD\_Type \* base )

Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
|-------------|--------------------------------------------------------------|

Returns

status flag; OR'ed value or the [\\_flexio\\_mculcd\\_status\\_flags](#).

Note

Don't use this function with DMA APIs.

### 23.6.7.5 void FLEXIO\_MCULCD\_ClearStatusFlags ( FLEXIO\_MCULCD\_Type \* base, uint32\_t mask )

Parameters

---

|             |                                                                                         |
|-------------|-----------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.                            |
| <i>mask</i> | Status to clear, it is the OR'ed value of <a href="#">_flexio_mculcd_status_flags</a> . |

Note

Don't use this function with DMA APIs.

### 23.6.7.6 void FLEXIO\_MCULCD\_EnableInterrupts ( FLEXIO\_MCULCD\_Type \* *base*, uint32\_t *mask* )

This function enables the FlexIO MCULCD interrupt.

Parameters

|             |                                                                                                  |
|-------------|--------------------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.                                     |
| <i>mask</i> | Interrupts to enable, it is the OR'ed value of <a href="#">_flexio_mculcd_interrupt_enable</a> . |

### 23.6.7.7 void FLEXIO\_MCULCD\_DisableInterrupts ( FLEXIO\_MCULCD\_Type \* *base*, uint32\_t *mask* )

This function disables the FlexIO MCULCD interrupt.

Parameters

|             |                                                                                                   |
|-------------|---------------------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.                                      |
| <i>mask</i> | Interrupts to disable, it is the OR'ed value of <a href="#">_flexio_mculcd_interrupt_enable</a> . |

### 23.6.7.8 static void FLEXIO\_MCULCD\_EnableTxDMA ( FLEXIO\_MCULCD\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                                              |
|---------------|--------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
| <i>enable</i> | True means enable DMA, false means disable DMA.              |

### 23.6.7.9 static void FLEXIO\_MCULCD\_EnableRxDMA ( FLEXIO\_MCULCD\_Type \* *base*, bool *enable* ) [inline], [static]



Parameters

|               |                                                              |
|---------------|--------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
| <i>enable</i> | True means enable DMA, false means disable DMA.              |

#### 23.6.7.10 `static uint32_t FLEXIO_MCULCD_GetTxDataRegisterAddress ( FLEXIO_MCULCD_Type * base ) [inline], [static]`

This function returns the MCULCD data register address, which is mainly used by DMA/eDMA.

Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
|-------------|--------------------------------------------------------------|

Returns

FlexIO MCULCD transmit data register address.

#### 23.6.7.11 `static uint32_t FLEXIO_MCULCD_GetRxDataRegisterAddress ( FLEXIO_MCULCD_Type * base ) [inline], [static]`

This function returns the MCULCD data register address, which is mainly used by DMA/eDMA.

Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
|-------------|--------------------------------------------------------------|

Returns

FlexIO MCULCD receive data register address.

#### 23.6.7.12 `status_t FLEXIO_MCULCD_SetBaudRate ( FLEXIO_MCULCD_Type * base, uint32_t baudRate_Bps, uint32_t srcClock_Hz )`

Parameters

|                     |                                                                  |
|---------------------|------------------------------------------------------------------|
| <i>base</i>         | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.     |
| <i>baudRate_Bps</i> | Desired baud rate in bit-per-second for all data lines combined. |
| <i>srcClock_Hz</i>  | FLEXIO clock frequency in Hz.                                    |

Return values

|                                |                              |
|--------------------------------|------------------------------|
| <i>kStatus_Success</i>         | Set successfully.            |
| <i>kStatus_InvalidArgument</i> | Could not set the baud rate. |

### 23.6.7.13 void FLEXIO\_MCULCD\_SetSingleBeatWriteConfig ( FLEXIO\_MCULCD\_Type \* *base* )

At the beginning multiple beats write operation, the FLEXIO MCULCD is configured to multiple beats write mode using this function. After write operation, the configuration is cleared by [FLEXIO\\_MCULCD\\_ClearSingleBeatWriteConfig](#).

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

Note

This is an internal used function, upper layer should not use.

### 23.6.7.14 void FLEXIO\_MCULCD\_ClearSingleBeatWriteConfig ( FLEXIO\_MCULCD\_Type \* *base* )

Clear the write configuration set by [FLEXIO\\_MCULCD\\_SetSingleBeatWriteConfig](#).

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

Note

This is an internal used function, upper layer should not use.

### 23.6.7.15 void FLEXIO\_MCULCD\_SetSingleBeatReadConfig ( FLEXIO\_MCULCD\_Type \* *base* )

At the beginning or multiple beats read operation, the FLEXIO MCULCD is configured to multiple beats read mode using this function. After read operation, the configuration is cleared by [FLEXIO\\_MCULCD-](#)

[\\_ClearSingleBeatReadConfig.](#)

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

Note

This is an internal used function, upper layer should not use.

### 23.6.7.16 void FLEXIO\_MCULCD\_ClearSingleBeatReadConfig ( FLEXIO\_MCULCD\_Type \* *base* )

Clear the read configuration set by [FLEXIO\\_MCULCD\\_SetSingleBeatReadConfig](#).

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

Note

This is an internal used function, upper layer should not use.

### 23.6.7.17 void FLEXIO\_MCULCD\_SetMultiBeatsWriteConfig ( FLEXIO\_MCULCD\_Type \* *base* )

At the beginning multiple beats write operation, the FLEXIO MCULCD is configured to multiple beats write mode using this function. After write operation, the configuration is cleared by [FLEXIO\\_MCULCD\\_ClearMultBeatsWriteConfig](#).

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

Note

This is an internal used function, upper layer should not use.

### 23.6.7.18 void FLEXIO\_MCULCD\_ClearMultiBeatsWriteConfig ( FLEXIO\_MCULCD\_Type \* *base* )

Clear the write configuration set by [FLEXIO\\_MCULCD\\_SetMultBeatsWriteConfig](#).

## Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

## Note

This is an internal used function, upper layer should not use.

### 23.6.7.19 void FLEXIO\_MCULCD\_SetMultiBeatsReadConfig ( FLEXIO\_MCULCD\_Type \* *base* )

At the begining or multiple beats read operation, the FLEXIO MCULCD is configured to multiple beats read mode using this function. After read operation, the configuration is cleared by FLEXIO\_MCULCD\_ClearMultiBeatsReadConfig.

## Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

## Note

This is an internal used function, upper layer should not use.

### 23.6.7.20 void FLEXIO\_MCULCD\_ClearMultiBeatsReadConfig ( FLEXIO\_MCULCD\_Type \* *base* )

Clear the read configuration set by FLEXIO\_MCULCD\_SetMultBeatsReadConfig.

## Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

## Note

This is an internal used function, upper layer should not use.

### 23.6.7.21 static void FLEXIO\_MCULCD\_Enable ( FLEXIO\_MCULCD\_Type \* *base*, bool *enable* ) [inline], [static]

## Parameters

|               |                                                     |
|---------------|-----------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
| <i>enable</i> | True to enable, false does not have any effect.     |

**23.6.7.22 uint32\_t FLEXIO\_MCULCD\_ReadData ( FLEXIO\_MCULCD\_Type \* base )**

Read data from the RX shift buffer directly, it does no check whether the buffer is empty or not.

If the data bus width is 8-bit:

```
* uint8_t value;
* value = (uint8_t)FLEXIO_MCULCD_ReadData(base);
*
```

If the data bus width is 16-bit:

```
* uint16_t value;
* value = (uint16_t)FLEXIO_MCULCD_ReadData(base);
*
```

## Note

This function returns the RX shifter buffer value (32-bit) directly. The return value should be converted according to data bus width.

## Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
|-------------|--------------------------------------------------------------|

## Returns

The data read out.

## Note

Don't use this function with DMA APIs.

**23.6.7.23 static void FLEXIO\_MCULCD\_WriteData ( FLEXIO\_MCULCD\_Type \* base, uint32\_t data ) [inline], [static]**

Write data into the TX shift buffer directly, it does no check whether the buffer is full or not.

## Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
| <i>data</i> | The data to write.                                           |

## Note

Don't use this function with DMA APIs.

**23.6.7.24** `static void FLEXIO_MCULCD_StartTransfer ( FLEXIO_MCULCD_Type * base ) [inline], [static]`

## Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
|-------------|--------------------------------------------------------------|

**23.6.7.25** `static void FLEXIO_MCULCD_StopTransfer ( FLEXIO_MCULCD_Type * base ) [inline], [static]`

## Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
|-------------|--------------------------------------------------------------|

**23.6.7.26** `void FLEXIO_MCULCD_WaitTransmitComplete ( void )`

Currently there is no effective method to wait for the data send out from the shiter, so here use a while loop to wait.

## Note

This is an internal used function.

**23.6.7.27** `void FLEXIO_MCULCD_WriteCommandBlocking ( FLEXIO_MCULCD_Type * base, uint32_t command )`

This function sends the command and returns when the command has been sent out.

Parameters

|                |                                                              |
|----------------|--------------------------------------------------------------|
| <i>base</i>    | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
| <i>command</i> | The command to send.                                         |

### 23.6.7.28 void FLEXIO\_MCULCD\_WriteDataArrayBlocking ( FLEXIO\_MCULCD\_Type \* *base*, const void \* *data*, size\_t *size* )

This function sends the data array and returns when the data sent out.

Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
| <i>data</i> | The data array to send.                                      |
| <i>size</i> | How many bytes to write.                                     |

### 23.6.7.29 void FLEXIO\_MCULCD\_ReadDataArrayBlocking ( FLEXIO\_MCULCD\_Type \* *base*, void \* *data*, size\_t *size* )

This function reads the data into array and returns when the data read finished.

Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
| <i>data</i> | The array to save the data.                                  |
| <i>size</i> | How many bytes to read.                                      |

### 23.6.7.30 void FLEXIO\_MCULCD\_WriteSameValueBlocking ( FLEXIO\_MCULCD\_Type \* *base*, uint32\_t *sameValue*, size\_t *size* )

This function sends the same value many times. It could be used to clear the LCD screen. If the data bus width is 8, this function will send LSB 8 bits of *sameValue* for *size* times. If the data bus is 16, this function will send LSB 16 bits of *sameValue* for *size* / 2 times.

Parameters



|                  |                                                              |
|------------------|--------------------------------------------------------------|
| <i>base</i>      | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
| <i>sameValue</i> | The same value to send.                                      |
| <i>size</i>      | How many bytes to send.                                      |

**23.6.7.31 void FLEXIO\_MCULCD\_TransferBlocking ( FLEXIO\_MCULCD\_Type \* *base*, flexio\_mculcd\_transfer\_t \* *xfer* )**

Note

The API does not return until the transfer finished.

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_MCULCD_Type</a> structure.       |
| <i>xfer</i> | pointer to <a href="#">flexio_mculcd_transfer_t</a> structure. |

**23.6.7.32 status\_t FLEXIO\_MCULCD\_TransferCreateHandle ( FLEXIO\_MCULCD\_Type \* *base*, flexio\_mculcd\_handle\_t \* *handle*, flexio\_mculcd\_transfer\_callback\_t *callback*, void \* *userData* )**

Parameters

|                 |                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------|
| <i>base</i>     | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.                                 |
| <i>handle</i>   | Pointer to the <a href="#">flexio_mculcd_handle_t</a> structure to store the transfer state. |
| <i>callback</i> | The callback function.                                                                       |
| <i>userData</i> | The parameter of the callback function.                                                      |

Return values

|                           |                                                |
|---------------------------|------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                |
| <i>kStatus_OutOfRange</i> | The FlexIO type/handle/ISR table out of range. |

**23.6.7.33 status\_t FLEXIO\_MCULCD\_TransferNonBlocking ( FLEXIO\_MCULCD\_Type \* *base*, flexio\_mculcd\_handle\_t \* *handle*, flexio\_mculcd\_transfer\_t \* *xfer* )**

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

## Parameters

|               |                                                                                           |
|---------------|-------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_mculcd_handle_t</code> structure to store the transfer state. |
| <i>xfer</i>   | FlexIO MCULCD transfer structure. See <a href="#">flexio_mculcd_transfer_t</a> .          |

## Return values

|                                   |                                       |
|-----------------------------------|---------------------------------------|
| <i>kStatus_Success</i>            | Successfully start a transfer.        |
| <i>kStatus_InvalidArgument</i>    | Input argument is invalid.            |
| <i>kStatus_FLEXIO_MCULCD_Busy</i> | MCULCD is busy with another transfer. |

**23.6.7.34** `void FLEXIO_MCULCD_TransferAbort ( FLEXIO_MCULCD_Type * base, flexio_mculcd_handle_t * handle )`

## Parameters

|               |                                                                                           |
|---------------|-------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_mculcd_handle_t</code> structure to store the transfer state. |

**23.6.7.35** `status_t FLEXIO_MCULCD_TransferGetCount ( FLEXIO_MCULCD_Type * base, flexio_mculcd_handle_t * handle, size_t * count )`

## Parameters

|               |                                                                                           |
|---------------|-------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_mculcd_handle_t</code> structure to store the transfer state. |
| <i>count</i>  | How many bytes transferred so far by the non-blocking transaction.                        |

## Return values

|                                     |                                         |
|-------------------------------------|-----------------------------------------|
| <i>kStatus_Success</i>              | Get the transferred count Successfully. |
| <i>kStatus_NoTransferInProgress</i> | No transfer in process.                 |

**23.6.7.36** `void FLEXIO_MCULCD_TransferHandleIRQ ( void * base, void * handle )`

### Parameters

|               |                                                                              |
|---------------|------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.                 |
| <i>handle</i> | Pointer to the flexio_mculcd_handle_t structure to store the transfer state. |

## 23.6.8 FlexIO eDMA MCU Interface LCD Driver

SDK provide eDMA transactional APIs to transfer data using eDMA, the eDMA method is similar with interrupt transactional method.

### 23.6.8.1 Overview

Note

eDMA transactional functions use multiple beats method for better performance, in contrast, the blocking functions and interrupt functions use single beat method. The function `FLEXIO_MCULCD_ReadData`, `FLEXIO_MCULCD_WriteData`, `FLEXIO_MCULCD_GetStatusFlags`, and `FLEXIO_MCULCD_ClearStatusFlags` are only used for single beat case, so don't use these functions to work together with eDMA functions.

## FlexIO eDMA MCU Interface LCD Driver

### FLEXIO LCD send/receive using a DMA method

```
flexio_MCULCD_edma_handle_t handle;
volatile bool completeFlag = false;
edma_handle_t rxEdmaHandle;
edma_handle_t txEdmaHandle;

void flexioLcdCallback(FLEXIO_MCULCD_Type *base, flexio_MCULCD_edma_handle_t *handle,
 status_t status, void *userData)
{
 if (kStatus_FLEXIO_MCULCD_Idle == status)
 {
 completeFlag = true;
 }
}

void main(void)
{
 // Create the edma Handle.
 EDMA_CreateHandle(&rxEdmaHandle, DMA0, channel);
 EDMA_CreateHandle(&txEdmaHandle, DMA0, channel);

 // Configure the DMAMUX.
 // ...
 // rxEdmaHandle should use the last FlexIO RX shifters as DMA request source.
 // txEdmaHandle should use the first FlexIO TX shifters as DMA request source.

 // Init the FlexIO LCD driver.
 FLEXIO_MCULCD_Init(...);

 // Create the transactional handle.
 FLEXIO_MCULCD_TransferCreateHandleEDMA(&flexioLcdDev, &handle,
 flexioLcdCallback, NULL, &txEdmaHandle, &rxEdmaHandle);

 xfer.command = command2;
 xfer.mode = kFLEXIO_MCULCD_WriteArray;
 xfer.dataAddrOrSameValue = (uint32_t)dataToSend;
 xfer.dataCount = sizeof(dataToSend);
 completeFlag = false;
 FLEXIO_MCULCD_TransferEDMA(&flexioLcdDev, &handle, &xfer);
}
```

```

while (!completeFlag)
{
}

xfer.command = command2;
xfer.mode = kFLEXIO_MCULCD_WriteSameValue;
xfer.dataAddrOrSameValue = value;
xfer.dataCount = 1000;
completeFlag = false;
FLEXIO_MCULCD_TransferEDMA(&flexioLcdDev, &handle, &xfer);

while (!completeFlag)
{
}

xfer.command = command3;
xfer.mode = kFLEXIO_MCULCD_ReadArray;
xfer.dataAddrOrSameValue = (uint32_t)dataToReceive;
xfer.dataCount = sizeof(dataToReceive);
completeFlag = false;
FLEXIO_MCULCD_TransferEDMA(&flexioLcdDev, &handle, &xfer);

while (!completeFlag)
{
}
}

```

## Data Structures

- struct [flexio\\_mculcd\\_edma\\_handle\\_t](#)  
*FlexIO MCULCD eDMA transfer handle, users should not touch the content of the handle. [More...](#)*

## Macros

- #define [FSL\\_FLEXIO\\_MCULCD\\_EDMA\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 4))  
*FlexIO MCULCD EDMA driver version.*

## Typedefs

- typedef void(\* [flexio\\_mculcd\\_edma\\_transfer\\_callback\\_t](#))(FLEXIO\_MCULCD\_Type \*base, flexio\_mculcd\_edma\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*FlexIO MCULCD master callback for transfer complete.*

## eDMA Transactional

- [status\\_t FLEXIO\\_MCULCD\\_TransferCreateHandleEDMA](#) (FLEXIO\_MCULCD\_Type \*base, flexio\_mculcd\_edma\_handle\_t \*handle, [flexio\\_mculcd\\_edma\\_transfer\\_callback\\_t](#) callback, void \*userData, [edma\\_handle\\_t](#) \*txDmaHandle, [edma\\_handle\\_t](#) \*rxDmaHandle)  
*Initializes the FLEXIO MCULCD master eDMA handle.*
- [status\\_t FLEXIO\\_MCULCD\\_TransferEDMA](#) (FLEXIO\_MCULCD\_Type \*base, flexio\_mculcd\_edma\_handle\_t \*handle, [flexio\\_mculcd\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking FlexIO MCULCD transfer using eDMA.*

- void `FLEXIO_MCULCD_TransferAbortEDMA` (`FLEXIO_MCULCD_Type *base`, `flexio_mculcd_edma_handle_t *handle`)  
*Aborts a FlexIO MCULCD transfer using eDMA.*
- `status_t FLEXIO_MCULCD_TransferGetCountEDMA` (`FLEXIO_MCULCD_Type *base`, `flexio_mculcd_edma_handle_t *handle`, `size_t *count`)  
*Gets the remaining bytes for FlexIO MCULCD eDMA transfer.*

## 23.6.8.2 Data Structure Documentation

### 23.6.8.2.1 struct flexio\_mculcd\_edma\_handle

typedef for `flexio_mculcd_edma_handle_t` in advance.

#### Data Fields

- `FLEXIO_MCULCD_Type * base`  
*Pointer to the `FLEXIO_MCULCD_Type`.*
- `uint8_t txShifterNum`  
*Number of shifters used for TX.*
- `uint8_t rxShifterNum`  
*Number of shifters used for RX.*
- `uint32_t minorLoopBytes`  
*eDMA transfer minor loop bytes.*
- `edma_modulo_t txEdmaModulo`  
*Modulo value for the FlexIO shifter buffer access.*
- `edma_modulo_t rxEdmaModulo`  
*Modulo value for the FlexIO shifter buffer access.*
- `uint32_t dataAddrOrSameValue`  
*When sending the same value for many times, this is the value to send.*
- `size_t dataCount`  
*Total count to be transferred.*
- `volatile size_t remainingCount`  
*Remaining count still not transferred.*
- `volatile uint32_t state`  
*FlexIO MCULCD driver internal state.*
- `edma_handle_t * txDmaHandle`  
*DMA handle for MCULCD TX.*
- `edma_handle_t * rxDmaHandle`  
*DMA handle for MCULCD RX.*
- `flexio_mculcd_edma_transfer_callback_t completionCallback`  
*Callback for MCULCD DMA transfer.*
- `void * userData`  
*User Data for MCULCD DMA callback.*

## Field Documentation

- (1) `FLEXIO_MCULCD_Type* flexio_mculcd_edma_handle_t::base`
- (2) `uint8_t flexio_mculcd_edma_handle_t::txShifterNum`
- (3) `uint8_t flexio_mculcd_edma_handle_t::rxShifterNum`
- (4) `uint32_t flexio_mculcd_edma_handle_t::minorLoopBytes`
- (5) `edma_modulo_t flexio_mculcd_edma_handle_t::txEdmaModulo`
- (6) `edma_modulo_t flexio_mculcd_edma_handle_t::rxEdmaModulo`
- (7) `uint32_t flexio_mculcd_edma_handle_t::dataAddrOrSameValue`

When writing or reading array, this is the address of the data array.

- (8) `size_t flexio_mculcd_edma_handle_t::dataCount`
- (9) `volatile size_t flexio_mculcd_edma_handle_t::remainingCount`
- (10) `volatile uint32_t flexio_mculcd_edma_handle_t::state`

### 23.6.8.3 Macro Definition Documentation

23.6.8.3.1 `#define FSL_FLEXIO_MCULCD_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))`

### 23.6.8.4 Typedef Documentation

23.6.8.4.1 `typedef void(* flexio_mculcd_edma_transfer_callback_t)(FLEXIO_MCULCD_Type *base, flexio_mculcd_edma_handle_t *handle, status_t status, void *userData)`

When transfer finished, the callback function is called and returns the `status` as `kStatus_FLEXIO_MCULCD_Idle`.

### 23.6.8.5 Function Documentation

23.6.8.5.1 `status_t FLEXIO_MCULCD_TransferCreateHandleEDMA ( FLEXIO_MCULCD_Type * base, flexio_mculcd_edma_handle_t * handle, flexio_mculcd_edma_transfer_callback_t callback, void * userData, edma_handle_t * txDmaHandle, edma_handle_t * rxDmaHandle )`

This function initializes the FLEXIO MCULCD master eDMA handle which can be used for other FLEXIO MCULCD transactional APIs. For a specified FLEXIO MCULCD instance, call this API once to get the initialized handle.

## Parameters

|                    |                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | Pointer to <a href="#">FLEXIO_MCULCD_Type</a> structure.                                                                        |
| <i>handle</i>      | Pointer to <code>flexio_mculcd_edma_handle_t</code> structure to store the transfer state.                                      |
| <i>callback</i>    | MCULCD transfer complete callback, NULL means no callback.                                                                      |
| <i>userData</i>    | callback function parameter.                                                                                                    |
| <i>txDmaHandle</i> | User requested eDMA handle for FlexIO MCULCD eDMA TX, the DMA request source of this handle should be the first of TX shifters. |
| <i>rxDmaHandle</i> | User requested eDMA handle for FlexIO MCULCD eDMA RX, the DMA request source of this handle should be the last of RX shifters.  |

## Return values

|                        |                                 |
|------------------------|---------------------------------|
| <i>kStatus_Success</i> | Successfully create the handle. |
|------------------------|---------------------------------|

### 23.6.8.5.2 `status_t FLEXIO_MCULCD_TransferEDMA ( FLEXIO_MCULCD_Type * base, flexio_mculcd_edma_handle_t * handle, flexio_mculcd_transfer_t * xfer )`

This function returns immediately after transfer initiates. To check whether the transfer is completed, user could:

1. Use the transfer completed callback;
2. Polling function `FLEXIO_MCULCD_GetTransferCountEDMA`

## Parameters

|               |                                                                                            |
|---------------|--------------------------------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_MCULCD_Type</a> structure.                                   |
| <i>handle</i> | pointer to <code>flexio_mculcd_edma_handle_t</code> structure to store the transfer state. |
| <i>xfer</i>   | Pointer to FlexIO MCULCD transfer structure.                                               |

## Return values

|                                   |                                                            |
|-----------------------------------|------------------------------------------------------------|
| <i>kStatus_Success</i>            | Successfully start a transfer.                             |
| <i>kStatus_InvalidArgument</i>    | Input argument is invalid.                                 |
| <i>kStatus_FLEXIO_MCULCD_Busy</i> | FlexIO MCULCD is not idle, it is running another transfer. |

### 23.6.8.5.3 `void FLEXIO_MCULCD_TransferAbortEDMA ( FLEXIO_MCULCD_Type * base, flexio_mculcd_edma_handle_t * handle )`



## Parameters

|               |                                                          |
|---------------|----------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_MCULCD_Type</a> structure. |
| <i>handle</i> | FlexIO MCULCD eDMA handle pointer.                       |

#### 23.6.8.5.4 `status_t FLEXIO_MCULCD_TransferGetCountEDMA ( FLEXIO_MCULCD_Type * base, flexio_mculcd_edma_handle_t * handle, size_t * count )`

## Parameters

|               |                                                             |
|---------------|-------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_MCULCD_Type</a> structure.    |
| <i>handle</i> | FlexIO MCULCD eDMA handle pointer.                          |
| <i>count</i>  | Number of count transferred so far by the eDMA transaction. |

## Return values

|                                     |                                         |
|-------------------------------------|-----------------------------------------|
| <i>kStatus_Success</i>              | Get the transferred count Successfully. |
| <i>kStatus_NoTransferInProgress</i> | No transfer in process.                 |

## 23.7 FlexIO SPI Driver

### 23.7.1 Overview

The MCUXpresso SDK provides a peripheral driver for an SPI function using the Flexible I/O module of MCUXpresso SDK devices.

FlexIO SPI driver includes functional APIs and transactional APIs.

Functional APIs target low-level APIs. Functional APIs can be used for FlexIO SPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FlexIO SPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the `FLEXIO_SPI_Type *base` as the first parameter. FlexIO SPI functional operation groups provide the functional API set.

Transactional APIs target high-level APIs. Transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs can satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the `flexio_spi_master_handle_t/flexio_spi_slave_handle_t` as the second parameter. Initialize the handle by calling the `FLEXIO_SPI_MasterTransferCreateHandle()` or `FLEXIO_SPI_SlaveTransferCreateHandle()` API.

Transactional APIs support asynchronous transfer. This means that the functions `FLEXIO_SPI_MasterTransferNonBlocking()/FLEXIO_SPI_SlaveTransferNonBlocking()` set up an interrupt for data transfer. When the transfer is complete, the upper layer is notified through a callback function with the `kStatus_FLEXIO_SPI_Idle` status.

Note that the FlexIO SPI slave driver only supports discontinuous PCS access, which is a limitation. The FlexIO SPI slave driver can support continuous PCS, but the slave cannot adapt discontinuous and continuous PCS automatically. Users can change the timer disable mode in `FLEXIO_SPI_SlaveInit` manually, from `kFLEXIO_TimerDisableOnTimerCompare` to `kFLEXIO_TimerDisableNever` to enable a discontinuous PCS access. Only `CPHA = 0` is supported.

### 23.7.2 Typical use case

#### 23.7.2.1 FlexIO SPI send/receive using an interrupt method

```
flexio_spi_master_handle_t g_spiHandle;
FLEXIO_SPI_Type spiDev;
volatile bool txFinished;
static uint8_t srcBuff[BUFFER_SIZE];
static uint8_t destBuff[BUFFER_SIZE];

void FLEXIO_SPI_MasterUserCallback(FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle
 , status_t status, void *userData)
{
 userData = userData;

 if (kStatus_FLEXIO_SPI_Idle == status)
 {
 txFinished = true;
 }
}
```

```

}

void main(void)
{
 //...
 flexio_spi_transfer_t xfer = {0};
 flexio_spi_master_config_t userConfig;

 FLEXIO_SPI_MasterGetDefaultConfig(&userConfig);
 userConfig.baudRate_Bps = 500000U;

 spiDev.flexioBase = BOARD_FLEXIO_BASE;
 spiDev.SDOPinIndex = FLEXIO_SPI_MOSI_PIN;
 spiDev.SDIPinIndex = FLEXIO_SPI_MISO_PIN;
 spiDev.SCKPinIndex = FLEXIO_SPI_SCK_PIN;
 spiDev.CSnPinIndex = FLEXIO_SPI_CSn_PIN;
 spiDev.shifterIndex[0] = 0U;
 spiDev.shifterIndex[1] = 1U;
 spiDev.timerIndex[0] = 0U;
 spiDev.timerIndex[1] = 1U;

 FLEXIO_SPI_MasterInit(&spiDev, &userConfig, FLEXIO_CLOCK_FREQUENCY);

 xfer.txData = srcBuff;
 xfer.rxData = destBuff;
 xfer.dataSize = BUFFER_SIZE;
 xfer.flags = kFLEXIO_SPI_8bitMsb;
 FLEXIO_SPI_MasterTransferCreateHandle(&spiDev, &g_spiHandle,
 FLEXIO_SPI_MasterUserCallback, NULL);
 FLEXIO_SPI_MasterTransferNonBlocking(&spiDev, &g_spiHandle, &xfer);

 // Send finished.
 while (!txFinished)
 {
 // ...
 }
}

```

### 23.7.2.2 FlexIO\_SPI Send/Receive in DMA way

```

dma_handle_t g_spiTxDmaHandle;
dma_handle_t g_spiRxDmaHandle;
flexio_spi_master_handle_t g_spiHandle;
FLEXIO_SPI_Type spiDev;
volatile bool txFinished;
static uint8_t srcBuff[BUFFER_SIZE];
static uint8_t destBuff[BUFFER_SIZE];
void FLEXIO_SPI_MasterUserCallback(FLEXIO_SPI_Type *base, flexio_spi_master_dma_handle_t
 *handle, status_t status, void *userData)
{
 userData = userData;

 if (kStatus_FLEXIO_SPI_Idle == status)
 {
 txFinished = true;
 }
}

void main(void)
{
 flexio_spi_transfer_t xfer = {0};
 flexio_spi_master_config_t userConfig;

 FLEXIO_SPI_MasterGetDefaultConfig(&userConfig);

```

```

userConfig.baudRate_Bps = 500000U;

spiDev.flexioBase = BOARD_FLEXIO_BASE;
spiDev.SDOPinIndex = FLEXIO_SPI_MOSI_PIN;
spiDev.SDIPinIndex = FLEXIO_SPI_MISO_PIN;
spiDev.SCKPinIndex = FLEXIO_SPI_SCK_PIN;
spiDev.CSnPinIndex = FLEXIO_SPI_CSn_PIN;
spiDev.shifterIndex[0] = 0U;
spiDev.shifterIndex[1] = 1U;
spiDev.timerIndex[0] = 0U;
spiDev.timerIndex[1] = 1U;

/* Init DMAMUX. */
DMAMUX_Init(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR)

/* Init the DMA/EDMA module */
#if defined(FSL_FEATURE_SOC_DMA_COUNT) && FSL_FEATURE_SOC_DMA_COUNT > 0U
DMA_Init(EXAMPLE_FLEXIO_SPI_DMA_BASEADDR);
DMA_CreateHandle(&txHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR, FLEXIO_SPI_TX_DMA_CHANNEL);
DMA_CreateHandle(&rxHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR, FLEXIO_SPI_RX_DMA_CHANNEL);
#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_FLEXIO_SPI_DMA_BASEADDR, &edmaConfig);
EDMA_CreateHandle(&txHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR,
FLEXIO_SPI_TX_DMA_CHANNEL);
EDMA_CreateHandle(&rxHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR,
FLEXIO_SPI_RX_DMA_CHANNEL);
#endif /* FSL_FEATURE_SOC_EDMA_COUNT */

dma_request_source_tx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + spiDev.
shifterIndex[0]);
dma_request_source_rx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + spiDev.
shifterIndex[1]);

/* Requests DMA channels for transmit and receive. */
DMAMUX_SetSource(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR, FLEXIO_SPI_TX_DMA_CHANNEL, (
dma_request_source_t)dma_request_source_tx);
DMAMUX_SetSource(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR, FLEXIO_SPI_RX_DMA_CHANNEL, (
dma_request_source_t)dma_request_source_rx);
DMAMUX_EnableChannel(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR,
FLEXIO_SPI_TX_DMA_CHANNEL);
DMAMUX_EnableChannel(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR,
FLEXIO_SPI_RX_DMA_CHANNEL);

FLEXIO_SPI_MasterInit(&spiDev, &userConfig, FLEXIO_CLOCK_FREQUENCY);

/* Initializes the buffer. */
for (i = 0; i < BUFFER_SIZE; i++)
{
 srcBuff[i] = i;
}

/* Sends to the slave. */
xfer.txData = srcBuff;
xfer.rxData = destBuff;
xfer.dataSize = BUFFER_SIZE;
xfer.flags = kFLEXIO_SPI_8bitMsb;
FLEXIO_SPI_MasterTransferCreateHandleDMA(&spiDev, &g_spiHandle, FLEXIO_SPI_MasterUserCallback, NULL
, &g_spiTxDmaHandle, &g_spiRxDmaHandle);
FLEXIO_SPI_MasterTransferDMA(&spiDev, &g_spiHandle, &xfer);

// Send finished.
while (!txFinished)
{

```

```

 }
 // ...
}

```

## Modules

- [FlexIO eDMA SPI Driver](#)

## Data Structures

- struct [FLEXIO\\_SPI\\_Type](#)  
*Define FlexIO SPI access structure typedef. [More...](#)*
- struct [flexio\\_spi\\_master\\_config\\_t](#)  
*Define FlexIO SPI master configuration structure. [More...](#)*
- struct [flexio\\_spi\\_slave\\_config\\_t](#)  
*Define FlexIO SPI slave configuration structure. [More...](#)*
- struct [flexio\\_spi\\_transfer\\_t](#)  
*Define FlexIO SPI transfer structure. [More...](#)*
- struct [flexio\\_spi\\_master\\_handle\\_t](#)  
*Define FlexIO SPI handle structure. [More...](#)*

## Macros

- #define [FLEXIO\\_SPI\\_DUMMYDATA](#) (0x00U)  
*FlexIO SPI dummy transfer data, the data is sent while txData is NULL.*
- #define [SPI\\_RETRY\\_TIMES](#) 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/  
*Retry times for waiting flag.*
- #define [FLEXIO\\_SPI\\_XFER\\_DATA\\_FORMAT](#)(flag) ((flag) & (0x7U))  
*Get the transfer data format of width and bit order.*

## Typedefs

- typedef [flexio\\_spi\\_master\\_handle\\_t](#) [flexio\\_spi\\_slave\\_handle\\_t](#)  
*Slave handle is the same with master handle.*
- typedef void(\* [flexio\\_spi\\_master\\_transfer\\_callback\\_t](#) )(FLEXIO\_SPI\_Type \*base, [flexio\\_spi\\_master\\_handle\\_t](#) \*handle, [status\\_t](#) status, void \*userData)  
*FlexIO SPI master callback for finished transmit.*
- typedef void(\* [flexio\\_spi\\_slave\\_transfer\\_callback\\_t](#) )(FLEXIO\_SPI\_Type \*base, [flexio\\_spi\\_slave\\_handle\\_t](#) \*handle, [status\\_t](#) status, void \*userData)  
*FlexIO SPI slave callback for finished transmit.*

## Enumerations

- enum {
  - kStatus\_FLEXIO\_SPI\_Busy = MAKE\_STATUS(kStatusGroup\_FLEXIO\_SPI, 1),
  - kStatus\_FLEXIO\_SPI\_Idle = MAKE\_STATUS(kStatusGroup\_FLEXIO\_SPI, 2),
  - kStatus\_FLEXIO\_SPI\_Error = MAKE\_STATUS(kStatusGroup\_FLEXIO\_SPI, 3),
  - kStatus\_FLEXIO\_SPI\_Timeout }

*Error codes for the FlexIO SPI driver.*
- enum flexio\_spi\_clock\_phase\_t {
  - kFLEXIO\_SPI\_ClockPhaseFirstEdge = 0x0U,
  - kFLEXIO\_SPI\_ClockPhaseSecondEdge = 0x1U }

*FlexIO SPI clock phase configuration.*
- enum flexio\_spi\_shift\_direction\_t {
  - kFLEXIO\_SPI\_MsbFirst = 0,
  - kFLEXIO\_SPI\_LsbFirst = 1 }

*FlexIO SPI data shifter direction options.*
- enum flexio\_spi\_data\_bitcount\_mode\_t {
  - kFLEXIO\_SPI\_8BitMode = 0x08U,
  - kFLEXIO\_SPI\_16BitMode = 0x10U,
  - kFLEXIO\_SPI\_32BitMode = 0x20U }

*FlexIO SPI data length mode options.*
- enum \_flexio\_spi\_interrupt\_enable {
  - kFLEXIO\_SPI\_TxEmptyInterruptEnable = 0x1U,
  - kFLEXIO\_SPI\_RxFullInterruptEnable = 0x2U }

*Define FlexIO SPI interrupt mask.*
- enum \_flexio\_spi\_status\_flags {
  - kFLEXIO\_SPI\_TxBufferEmptyFlag = 0x1U,
  - kFLEXIO\_SPI\_RxBufferFullFlag = 0x2U }

*Define FlexIO SPI status mask.*
- enum \_flexio\_spi\_dma\_enable {
  - kFLEXIO\_SPI\_TxDmaEnable = 0x1U,
  - kFLEXIO\_SPI\_RxDmaEnable = 0x2U,
  - kFLEXIO\_SPI\_DmaAllEnable = 0x3U }

*Define FlexIO SPI DMA mask.*
- enum \_flexio\_spi\_transfer\_flags {
  - kFLEXIO\_SPI\_8bitMsb = 0x0U,
  - kFLEXIO\_SPI\_8bitLsb = 0x1U,
  - kFLEXIO\_SPI\_16bitMsb = 0x2U,
  - kFLEXIO\_SPI\_16bitLsb = 0x3U,
  - kFLEXIO\_SPI\_32bitMsb = 0x4U,
  - kFLEXIO\_SPI\_32bitLsb = 0x5U,
  - kFLEXIO\_SPI\_csContinuous = 0x8U }

*Define FlexIO SPI transfer flags.*

## Driver version

- #define `FSL_FLEXIO_SPI_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 2)`)  
*FlexIO SPI driver version.*

## FlexIO SPI Configuration

- void `FLEXIO_SPI_MasterInit` (`FLEXIO_SPI_Type *base`, `flexio_spi_master_config_t *masterConfig`, `uint32_t srcClock_Hz`)  
*Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration.*
- void `FLEXIO_SPI_MasterDeinit` (`FLEXIO_SPI_Type *base`)  
*Resets the FlexIO SPI timer and shifter config.*
- void `FLEXIO_SPI_MasterGetDefaultConfig` (`flexio_spi_master_config_t *masterConfig`)  
*Gets the default configuration to configure the FlexIO SPI master.*
- void `FLEXIO_SPI_SlaveInit` (`FLEXIO_SPI_Type *base`, `flexio_spi_slave_config_t *slaveConfig`)  
*Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration.*
- void `FLEXIO_SPI_SlaveDeinit` (`FLEXIO_SPI_Type *base`)  
*Gates the FlexIO clock.*
- void `FLEXIO_SPI_SlaveGetDefaultConfig` (`flexio_spi_slave_config_t *slaveConfig`)  
*Gets the default configuration to configure the FlexIO SPI slave.*

## Status

- `uint32_t FLEXIO_SPI_GetStatusFlags` (`FLEXIO_SPI_Type *base`)  
*Gets FlexIO SPI status flags.*
- void `FLEXIO_SPI_ClearStatusFlags` (`FLEXIO_SPI_Type *base`, `uint32_t mask`)  
*Clears FlexIO SPI status flags.*

## Interrupts

- void `FLEXIO_SPI_EnableInterrupts` (`FLEXIO_SPI_Type *base`, `uint32_t mask`)  
*Enables the FlexIO SPI interrupt.*
- void `FLEXIO_SPI_DisableInterrupts` (`FLEXIO_SPI_Type *base`, `uint32_t mask`)  
*Disables the FlexIO SPI interrupt.*

## DMA Control

- void `FLEXIO_SPI_EnableDMA` (`FLEXIO_SPI_Type *base`, `uint32_t mask`, `bool enable`)  
*Enables/disables the FlexIO SPI transmit DMA.*
- static `uint32_t FLEXIO_SPI_GetTxDataRegisterAddress` (`FLEXIO_SPI_Type *base`, `flexio_spi_shift_direction_t direction`)  
*Gets the FlexIO SPI transmit data register address for MSB first transfer.*

- static uint32\_t `FLEXIO_SPI_GetRxDataRegisterAddress` (`FLEXIO_SPI_Type *base`, `flexio_spi_shift_direction_t direction`)  
*Gets the FlexIO SPI receive data register address for the MSB first transfer.*

## Bus Operations

- static void `FLEXIO_SPI_Enable` (`FLEXIO_SPI_Type *base`, bool enable)  
*Enables/disables the FlexIO SPI module operation.*
- void `FLEXIO_SPI_MasterSetBaudRate` (`FLEXIO_SPI_Type *base`, uint32\_t baudRate\_Bps, uint32\_t srcClockHz)  
*Sets baud rate for the FlexIO SPI transfer, which is only used for the master.*
- static void `FLEXIO_SPI_WriteData` (`FLEXIO_SPI_Type *base`, `flexio_spi_shift_direction_t direction`, uint32\_t data)  
*Writes one byte of data, which is sent using the MSB method.*
- static uint32\_t `FLEXIO_SPI_ReadData` (`FLEXIO_SPI_Type *base`, `flexio_spi_shift_direction_t direction`)  
*Reads 8 bit/16 bit data.*
- `status_t FLEXIO_SPI_WriteBlocking` (`FLEXIO_SPI_Type *base`, `flexio_spi_shift_direction_t direction`, const uint8\_t \*buffer, size\_t size)  
*Sends a buffer of data bytes.*
- `status_t FLEXIO_SPI_ReadBlocking` (`FLEXIO_SPI_Type *base`, `flexio_spi_shift_direction_t direction`, uint8\_t \*buffer, size\_t size)  
*Receives a buffer of bytes.*
- `status_t FLEXIO_SPI_MasterTransferBlocking` (`FLEXIO_SPI_Type *base`, `flexio_spi_transfer_t *xfer`)  
*Receives a buffer of bytes.*
- void `FLEXIO_SPI_FlushShifters` (`FLEXIO_SPI_Type *base`)  
*Flush tx/rx shifters.*

## Transactional

- `status_t FLEXIO_SPI_MasterTransferCreateHandle` (`FLEXIO_SPI_Type *base`, `flexio_spi_master_handle_t *handle`, `flexio_spi_master_transfer_callback_t callback`, void \*userData)  
*Initializes the FlexIO SPI Master handle, which is used in transactional functions.*
- `status_t FLEXIO_SPI_MasterTransferNonBlocking` (`FLEXIO_SPI_Type *base`, `flexio_spi_master_handle_t *handle`, `flexio_spi_transfer_t *xfer`)  
*Master transfer data using IRQ.*
- void `FLEXIO_SPI_MasterTransferAbort` (`FLEXIO_SPI_Type *base`, `flexio_spi_master_handle_t *handle`)  
*Aborts the master data transfer, which used IRQ.*
- `status_t FLEXIO_SPI_MasterTransferGetCount` (`FLEXIO_SPI_Type *base`, `flexio_spi_master_handle_t *handle`, size\_t \*count)  
*Gets the data transfer status which used IRQ.*
- void `FLEXIO_SPI_MasterTransferHandleIRQ` (void \*spiType, void \*spiHandle)  
*FlexIO SPI master IRQ handler function.*
- `status_t FLEXIO_SPI_SlaveTransferCreateHandle` (`FLEXIO_SPI_Type *base`, `flexio_spi_slave_handle_t *handle`, `flexio_spi_slave_transfer_callback_t callback`, void \*userData)



- Initializes the FlexIO SPI Slave handle, which is used in transactional functions.*

  - `status_t FLEXIO_SPI_SlaveTransferNonBlocking` (`FLEXIO_SPI_Type *base`, `flexio_spi_slave_handle_t *handle`, `flexio_spi_transfer_t *xfer`)
    - Slave transfer data using IRQ.*
  - `static void FLEXIO_SPI_SlaveTransferAbort` (`FLEXIO_SPI_Type *base`, `flexio_spi_slave_handle_t *handle`)
    - Aborts the slave data transfer which used IRQ, share same API with master.*
  - `static status_t FLEXIO_SPI_SlaveTransferGetCount` (`FLEXIO_SPI_Type *base`, `flexio_spi_slave_handle_t *handle`, `size_t *count`)
    - Gets the data transfer status which used IRQ, share same API with master.*
  - `void FLEXIO_SPI_SlaveTransferHandleIRQ` (`void *spiType`, `void *spiHandle`)
    - FlexIO SPI slave IRQ handler function.*

### 23.7.3 Data Structure Documentation

#### 23.7.3.1 struct FLEXIO\_SPI\_Type

##### Data Fields

- `FLEXIO_Type * flexioBase`
  - FlexIO base pointer.*
- `uint8_t SDOPinIndex`
  - Pin select for data output.*
- `uint8_t SDIPinIndex`
  - Pin select for data input.*
- `uint8_t SCKPinIndex`
  - Pin select for clock.*
- `uint8_t CSnPinIndex`
  - Pin select for enable.*
- `uint8_t shifterIndex` [2]
  - Shifter index used in FlexIO SPI.*
- `uint8_t timerIndex` [2]
  - Timer index used in FlexIO SPI.*

##### Field Documentation

(1) `FLEXIO_Type* FLEXIO_SPI_Type::flexioBase`

(2) `uint8_t FLEXIO_SPI_Type::SDOPinIndex`

To set SDO pin in Hi-Z state, user needs to mux the pin as GPIO input and disable all pull up/down in application.

- (3) `uint8_t FLEXIO_SPI_Type::SDIPinIndex`
- (4) `uint8_t FLEXIO_SPI_Type::SCKPinIndex`
- (5) `uint8_t FLEXIO_SPI_Type::CSnPinIndex`
- (6) `uint8_t FLEXIO_SPI_Type::shifterIndex[2]`
- (7) `uint8_t FLEXIO_SPI_Type::timerIndex[2]`

### 23.7.3.2 struct flexio\_spi\_master\_config\_t

#### Data Fields

- `bool enableMaster`  
*Enable/disable FlexIO SPI master after configuration.*
- `bool enableInDoze`  
*Enable/disable FlexIO operation in doze mode.*
- `bool enableInDebug`  
*Enable/disable FlexIO operation in debug mode.*
- `bool enableFastAccess`  
*Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- `uint32_t baudRate_Bps`  
*Baud rate in Bps.*
- `flexio_spi_clock_phase_t phase`  
*Clock phase.*
- `flexio_spi_data_bitcount_mode_t dataMode`  
*8bit or 16bit mode.*

#### Field Documentation

- (1) `bool flexio_spi_master_config_t::enableMaster`
- (2) `bool flexio_spi_master_config_t::enableInDoze`
- (3) `bool flexio_spi_master_config_t::enableInDebug`
- (4) `bool flexio_spi_master_config_t::enableFastAccess`
- (5) `uint32_t flexio_spi_master_config_t::baudRate_Bps`
- (6) `flexio_spi_clock_phase_t flexio_spi_master_config_t::phase`
- (7) `flexio_spi_data_bitcount_mode_t flexio_spi_master_config_t::dataMode`

### 23.7.3.3 struct flexio\_spi\_slave\_config\_t

#### Data Fields

- `bool enableSlave`

- *Enable/disable FlexIO SPI slave after configuration.*
- bool `enableInDoze`  
*Enable/disable FlexIO operation in doze mode.*
- bool `enableInDebug`  
*Enable/disable FlexIO operation in debug mode.*
- bool `enableFastAccess`  
*Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- `flexio_spi_clock_phase_t` `phase`  
*Clock phase.*
- `flexio_spi_data_bitcount_mode_t` `dataMode`  
*8bit or 16bit mode.*

### Field Documentation

- (1) `bool flexio_spi_slave_config_t::enableSlave`
- (2) `bool flexio_spi_slave_config_t::enableInDoze`
- (3) `bool flexio_spi_slave_config_t::enableInDebug`
- (4) `bool flexio_spi_slave_config_t::enableFastAccess`
- (5) `flexio_spi_clock_phase_t flexio_spi_slave_config_t::phase`
- (6) `flexio_spi_data_bitcount_mode_t flexio_spi_slave_config_t::dataMode`

### 23.7.3.4 struct flexio\_spi\_transfer\_t

#### Data Fields

- `uint8_t * txData`  
*Send buffer.*
- `uint8_t * rxData`  
*Receive buffer.*
- `size_t dataSize`  
*Transfer bytes.*
- `uint8_t flags`  
*FlexIO SPI control flag, MSB first or LSB first.*

## Field Documentation

- (1) `uint8_t* flexio_spi_transfer_t::txData`
- (2) `uint8_t* flexio_spi_transfer_t::rxData`
- (3) `size_t flexio_spi_transfer_t::dataSize`
- (4) `uint8_t flexio_spi_transfer_t::flags`

### 23.7.3.5 struct `_flexio_spi_master_handle`

typedef for `flexio_spi_master_handle_t` in advance.

## Data Fields

- `uint8_t * txData`  
*Transfer buffer.*
- `uint8_t * rxData`  
*Receive buffer.*
- `size_t transferSize`  
*Total bytes to be transferred.*
- `volatile size_t txRemainingBytes`  
*Send data remaining in bytes.*
- `volatile size_t rxRemainingBytes`  
*Receive data remaining in bytes.*
- `volatile uint32_t state`  
*FlexIO SPI internal state.*
- `uint8_t bytePerFrame`  
*SPI mode, 2bytes or 1byte in a frame.*
- `flexio_spi_shift_direction_t direction`  
*Shift direction.*
- `flexio_spi_master_transfer_callback_t callback`  
*FlexIO SPI callback.*
- `void * userData`  
*Callback parameter.*

## Field Documentation

- (1) `uint8_t* flexio_spi_master_handle_t::txData`
- (2) `uint8_t* flexio_spi_master_handle_t::rxData`
- (3) `size_t flexio_spi_master_handle_t::transferSize`
- (4) `volatile size_t flexio_spi_master_handle_t::txRemainingBytes`
- (5) `volatile size_t flexio_spi_master_handle_t::rxRemainingBytes`
- (6) `volatile uint32_t flexio_spi_master_handle_t::state`
- (7) `flexio_spi_shift_direction_t flexio_spi_master_handle_t::direction`
- (8) `flexio_spi_master_transfer_callback_t flexio_spi_master_handle_t::callback`
- (9) `void* flexio_spi_master_handle_t::userData`

## 23.7.4 Macro Definition Documentation

23.7.4.1 `#define FSL_FLEXIO_SPI_DRIVER_VERSION (MAKE_VERSION(2, 3, 2))`

23.7.4.2 `#define FLEXIO_SPI_DUMMYDATA (0x00U)`

23.7.4.3 `#define SPI_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`

23.7.4.4 `#define FLEXIO_SPI_XFER_DATA_FORMAT( flag ) ((flag) & (0x7U))`

## 23.7.5 Typedef Documentation

23.7.5.1 `typedef flexio_spi_master_handle_t flexio_spi_slave_handle_t`

## 23.7.6 Enumeration Type Documentation

### 23.7.6.1 anonymous enum

Enumerator

*kStatus\_FLEXIO\_SPI\_Busy* FlexIO SPI is busy.

*kStatus\_FLEXIO\_SPI\_Idle* SPI is idle.

*kStatus\_FLEXIO\_SPI\_Error* FlexIO SPI error.

*kStatus\_FLEXIO\_SPI\_Timeout* FlexIO SPI timeout polling status flags.

**23.7.6.2 enum flexio\_spi\_clock\_phase\_t**

Enumerator

*kFLEXIO\_SPI\_ClockPhaseFirstEdge* First edge on SPSCCK occurs at the middle of the first cycle of a data transfer.

*kFLEXIO\_SPI\_ClockPhaseSecondEdge* First edge on SPSCCK occurs at the start of the first cycle of a data transfer.

**23.7.6.3 enum flexio\_spi\_shift\_direction\_t**

Enumerator

*kFLEXIO\_SPI\_MsbFirst* Data transfers start with most significant bit.

*kFLEXIO\_SPI\_LsbFirst* Data transfers start with least significant bit.

**23.7.6.4 enum flexio\_spi\_data\_bitcount\_mode\_t**

Enumerator

*kFLEXIO\_SPI\_8BitMode* 8-bit data transmission mode.

*kFLEXIO\_SPI\_16BitMode* 16-bit data transmission mode.

*kFLEXIO\_SPI\_32BitMode* 32-bit data transmission mode.

**23.7.6.5 enum \_flexio\_spi\_interrupt\_enable**

Enumerator

*kFLEXIO\_SPI\_TxEmptyInterruptEnable* Transmit buffer empty interrupt enable.

*kFLEXIO\_SPI\_RxFullInterruptEnable* Receive buffer full interrupt enable.

**23.7.6.6 enum \_flexio\_spi\_status\_flags**

Enumerator

*kFLEXIO\_SPI\_TxBufferEmptyFlag* Transmit buffer empty flag.

*kFLEXIO\_SPI\_RxBufferFullFlag* Receive buffer full flag.

### 23.7.6.7 enum \_flexio\_spi\_dma\_enable

Enumerator

*kFLEXIO\_SPI\_TxDmaEnable* Tx DMA request source.  
*kFLEXIO\_SPI\_RxDmaEnable* Rx DMA request source.  
*kFLEXIO\_SPI\_DmaAllEnable* All DMA request source.

### 23.7.6.8 enum \_flexio\_spi\_transfer\_flags

Note

Use *kFLEXIO\_SPI\_csContinuous* and one of the other flags to OR together to form the transfer flag.

Enumerator

*kFLEXIO\_SPI\_8bitMsb* FlexIO SPI 8-bit MSB first.  
*kFLEXIO\_SPI\_8bitLsb* FlexIO SPI 8-bit LSB first.  
*kFLEXIO\_SPI\_16bitMsb* FlexIO SPI 16-bit MSB first.  
*kFLEXIO\_SPI\_16bitLsb* FlexIO SPI 16-bit LSB first.  
*kFLEXIO\_SPI\_32bitMsb* FlexIO SPI 32-bit MSB first.  
*kFLEXIO\_SPI\_32bitLsb* FlexIO SPI 32-bit LSB first.  
*kFLEXIO\_SPI\_csContinuous* Enable the CS signal continuous mode.

## 23.7.7 Function Documentation

### 23.7.7.1 void FLEXIO\_SPI\_MasterInit ( FLEXIO\_SPI\_Type \* *base*, flexio\_spi\_master\_config\_t \* *masterConfig*, uint32\_t *srcClock\_Hz* )

The configuration structure can be filled by the user, or be set with default values by the [FLEXIO\\_SPI\\_MasterGetDefaultConfig\(\)](#).

Note

1.FlexIO SPI master only support CPOL = 0, which means clock inactive low. 2.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI master communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by  $2*2=4$ . If FlexIO SPI master communicates with FlexIO SPI slave, the maximum baud rate is FlexIO clock frequency divided by  $(1.5+2.5)*2=8$ .

Example

```
FLEXIO_SPI_Type spiDev = {
 .flexioBase = FLEXIO,
 .SDOPinIndex = 0,
 .SDIPinIndex = 1,
```

```

.SCKPinIndex = 2,
.CSnPinIndex = 3,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_spi_master_config_t config = {
.enableMaster = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 500000,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_MasterInit(&spiDev, &config, srcClock_Hz);

```

## Parameters

|                     |                                                                      |
|---------------------|----------------------------------------------------------------------|
| <i>base</i>         | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.            |
| <i>masterConfig</i> | Pointer to the <a href="#">flexio_spi_master_config_t</a> structure. |
| <i>srcClock_Hz</i>  | FlexIO source clock in Hz.                                           |

### 23.7.7.2 void FLEXIO\_SPI\_MasterDeinit ( FLEXIO\_SPI\_Type \* *base* )

## Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_SPI_Type</a> . |
|-------------|--------------------------------------------------|

### 23.7.7.3 void FLEXIO\_SPI\_MasterGetDefaultConfig ( flexio\_spi\_master\_config\_t \* *masterConfig* )

The configuration can be used directly by calling the [FLEXIO\\_SPI\\_MasterConfigure\(\)](#). Example:

```

flexio_spi_master_config_t masterConfig;
FLEXIO_SPI_MasterGetDefaultConfig(&masterConfig);

```

## Parameters

|                     |                                                                      |
|---------------------|----------------------------------------------------------------------|
| <i>masterConfig</i> | Pointer to the <a href="#">flexio_spi_master_config_t</a> structure. |
|---------------------|----------------------------------------------------------------------|

### 23.7.7.4 void FLEXIO\_SPI\_SlaveInit ( FLEXIO\_SPI\_Type \* *base*, flexio\_spi\_slave\_config\_t \* *slaveConfig* )

The configuration structure can be filled by the user, or be set with default values by the [FLEXIO\\_SPI\\_SlaveGetDefaultConfig\(\)](#).



## Note

1. Only one timer is needed in the FlexIO SPI slave. As a result, the second timer index is ignored. 2.- FlexIO SPI slave only support CPOL = 0, which means clock inactive low. 3. For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI slave communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by  $3*2=6$ . If FlexIO SPI slave communicates with FlexIO SPI master, the maximum baud rate is FlexIO clock frequency divided by  $(1.5+2.5)*2=8$ . Example

```
FLEXIO_SPI_Type spiDev = {
 .flexioBase = FLEXIO,
 .SDOPinIndex = 0,
 .SDIPinIndex = 1,
 .SCKPinIndex = 2,
 .CSnPinIndex = 3,
 .shifterIndex = {0,1},
 .timerIndex = {0}
};
flexio_spi_slave_config_t config = {
 .enableSlave = true,
 .enableInDoze = false,
 .enableInDebug = true,
 .enableFastAccess = false,
 .phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
 .direction = kFLEXIO_SPI_MsbFirst,
 .dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_SlaveInit(&spiDev, &config);
```

## Parameters

|                    |                                                                     |
|--------------------|---------------------------------------------------------------------|
| <i>base</i>        | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.           |
| <i>slaveConfig</i> | Pointer to the <a href="#">flexio_spi_slave_config_t</a> structure. |

### 23.7.7.5 void FLEXIO\_SPI\_SlaveDeinit ( FLEXIO\_SPI\_Type \* *base* )

## Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_SPI_Type</a> . |
|-------------|--------------------------------------------------|

### 23.7.7.6 void FLEXIO\_SPI\_SlaveGetDefaultConfig ( flexio\_spi\_slave\_config\_t \* *slaveConfig* )

The configuration can be used directly for calling the FLEXIO\_SPI\_SlaveConfigure(). Example:

```
flexio_spi_slave_config_t slaveConfig;
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);
```

## Parameters

|                    |                                                                     |
|--------------------|---------------------------------------------------------------------|
| <i>slaveConfig</i> | Pointer to the <a href="#">flexio_spi_slave_config_t</a> structure. |
|--------------------|---------------------------------------------------------------------|

**23.7.7.7 uint32\_t FLEXIO\_SPI\_GetStatusFlags ( FLEXIO\_SPI\_Type \* *base* )**

## Parameters

|             |                                                           |
|-------------|-----------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
|-------------|-----------------------------------------------------------|

## Returns

status flag; Use the status flag to AND the following flag mask and get the status.

- kFLEXIO\_SPI\_TxEmptyFlag
- kFLEXIO\_SPI\_RxEmptyFlag

**23.7.7.8 void FLEXIO\_SPI\_ClearStatusFlags ( FLEXIO\_SPI\_Type \* *base*, uint32\_t *mask* )**

## Parameters

|             |                                                                                                                                                                                          |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                                                                                                                |
| <i>mask</i> | status flag The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• kFLEXIO_SPI_TxEmptyFlag</li> <li>• kFLEXIO_SPI_RxEmptyFlag</li> </ul> |

**23.7.7.9 void FLEXIO\_SPI\_EnableInterrupts ( FLEXIO\_SPI\_Type \* *base*, uint32\_t *mask* )**

This function enables the FlexIO SPI interrupt.

## Parameters

|             |                                                           |
|-------------|-----------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
|-------------|-----------------------------------------------------------|

|             |                                                                                                                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>mask</i> | interrupt source. The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• kFLEXIO_SPI_RxFullInterruptEnable</li> <li>• kFLEXIO_SPI_TxEmptyInterruptEnable</li> </ul> |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 23.7.7.10 void FLEXIO\_SPI\_DisableInterrupts ( FLEXIO\_SPI\_Type \* *base*, uint32\_t *mask* )

This function disables the FlexIO SPI interrupt.

Parameters

|             |                                                                                                                                                                                                                    |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                                                                                                                                          |
| <i>mask</i> | interrupt source The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• kFLEXIO_SPI_RxFullInterruptEnable</li> <li>• kFLEXIO_SPI_TxEmptyInterruptEnable</li> </ul> |

### 23.7.7.11 void FLEXIO\_SPI\_EnableDMA ( FLEXIO\_SPI\_Type \* *base*, uint32\_t *mask*, bool *enable* )

This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the kFLEXIO\_SPI\_TxEmptyFlag does/doesn't trigger the DMA request.

Parameters

|               |                                                           |
|---------------|-----------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>mask</i>   | SPI DMA source.                                           |
| <i>enable</i> | True means enable DMA, false means disable DMA.           |

### 23.7.7.12 static uint32\_t FLEXIO\_SPI\_GetTxDataRegisterAddress ( FLEXIO\_SPI\_Type \* *base*, flexio\_spi\_shift\_direction\_t *direction* ) [inline], [static]

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>direction</i> | Shift direction of MSB first or LSB first.                |

Returns

FlexIO SPI transmit data register address.

**23.7.7.13** `static uint32_t FLEXIO_SPI_GetRxDataRegisterAddress ( FLEXIO_SPI_Type *  
base, flexio_spi_shift_direction_t direction ) [inline], [static]`

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>direction</i> | Shift direction of MSB first or LSB first.                |

Returns

FlexIO SPI receive data register address.

**23.7.7.14** `static void FLEXIO_SPI_Enable ( FLEXIO_SPI_Type * base, bool enable )  
[inline], [static]`

Parameters

|               |                                                  |
|---------------|--------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> . |
| <i>enable</i> | True to enable, false does not have any effect.  |

**23.7.7.15** `void FLEXIO_SPI_MasterSetBaudRate ( FLEXIO_SPI_Type * base, uint32_t  
baudRate_Bps, uint32_t srcClockHz )`

Parameters

|                     |                                                           |
|---------------------|-----------------------------------------------------------|
| <i>base</i>         | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>baudRate_Bps</i> | Baud Rate needed in Hz.                                   |
| <i>srcClockHz</i>   | SPI source clock frequency in Hz.                         |

**23.7.7.16** `static void FLEXIO_SPI_WriteData ( FLEXIO_SPI_Type * base,  
flexio_spi_shift_direction_t direction, uint32_t data ) [inline], [static]`

## Note

This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

## Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>direction</i> | Shift direction of MSB first or LSB first.                |
| <i>data</i>      | 8/16/32 bit data.                                         |

**23.7.7.17** `static uint32_t FLEXIO_SPI_ReadData ( FLEXIO_SPI_Type * base, flexio_spi_shift_direction_t direction ) [inline], [static]`

## Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

## Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>direction</i> | Shift direction of MSB first or LSB first.                |

## Returns

8 bit/16 bit data received.

**23.7.7.18** `status_t FLEXIO_SPI_WriteBlocking ( FLEXIO_SPI_Type * base, flexio_spi_shift_direction_t direction, const uint8_t * buffer, size_t size )`

## Note

This function blocks using the polling method until all bytes have been sent.

## Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>direction</i> | Shift direction of MSB first or LSB first.                |
| <i>buffer</i>    | The data bytes to send.                                   |
| <i>size</i>      | The number of data bytes to send.                         |

Return values

|                                   |                                         |
|-----------------------------------|-----------------------------------------|
| <i>kStatus_Success</i>            | Successfully create the handle.         |
| <i>kStatus_FLEXIO_SPI_Timeout</i> | The transfer timed out and was aborted. |

### 23.7.7.19 `status_t FLEXIO_SPI_ReadBlocking ( FLEXIO_SPI_Type * base, flexio_spi_shift_direction_t direction, uint8_t * buffer, size_t size )`

Note

This function blocks using the polling method until all bytes have been received.

Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>direction</i> | Shift direction of MSB first or LSB first.                |
| <i>buffer</i>    | The buffer to store the received bytes.                   |
| <i>size</i>      | The number of data bytes to be received.                  |
| <i>direction</i> | Shift direction of MSB first or LSB first.                |

Return values

|                                   |                                         |
|-----------------------------------|-----------------------------------------|
| <i>kStatus_Success</i>            | Successfully create the handle.         |
| <i>kStatus_FLEXIO_SPI_Timeout</i> | The transfer timed out and was aborted. |

### 23.7.7.20 `status_t FLEXIO_SPI_MasterTransferBlocking ( FLEXIO_SPI_Type * base, flexio_spi_transfer_t * xfer )`

Note

This function blocks via polling until all bytes have been received.

## Parameters

|             |                                                                            |
|-------------|----------------------------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_SPI_Type</a> structure                       |
| <i>xfer</i> | FlexIO SPI transfer structure, see <a href="#">flexio_spi_transfer_t</a> . |

## Return values

|                                   |                                         |
|-----------------------------------|-----------------------------------------|
| <i>kStatus_Success</i>            | Successfully create the handle.         |
| <i>kStatus_FLEXIO_SPI-Timeout</i> | The transfer timed out and was aborted. |

**23.7.7.21 void FLEXIO\_SPI\_FlushShifters ( FLEXIO\_SPI\_Type \* base )**

## Parameters

|             |                                                           |
|-------------|-----------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
|-------------|-----------------------------------------------------------|

**23.7.7.22 status\_t FLEXIO\_SPI\_MasterTransferCreateHandle ( FLEXIO\_SPI\_Type \* base, flexio\_spi\_master\_handle\_t \* handle, flexio\_spi\_master\_transfer\_callback\_t callback, void \* userData )**

## Parameters

|                 |                                                                                                  |
|-----------------|--------------------------------------------------------------------------------------------------|
| <i>base</i>     | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                        |
| <i>handle</i>   | Pointer to the <a href="#">flexio_spi_master_handle_t</a> structure to store the transfer state. |
| <i>callback</i> | The callback function.                                                                           |
| <i>userData</i> | The parameter of the callback function.                                                          |

## Return values

|                           |                                                |
|---------------------------|------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                |
| <i>kStatus_OutOfRange</i> | The FlexIO type/handle/ISR table out of range. |

**23.7.7.23 status\_t FLEXIO\_SPI\_MasterTransferNonBlocking ( FLEXIO\_SPI\_Type \* base, flexio\_spi\_master\_handle\_t \* handle, flexio\_spi\_transfer\_t \* xfer )**

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

## Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                     |
| <i>handle</i> | Pointer to the <code>flexio_spi_master_handle_t</code> structure to store the transfer state. |
| <i>xfer</i>   | FlexIO SPI transfer structure. See <a href="#">flexio_spi_transfer_t</a> .                    |

## Return values

|                                |                                               |
|--------------------------------|-----------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                    |
| <i>kStatus_FLEXIO_SPI_Busy</i> | SPI is not idle, is running another transfer. |

**23.7.7.24** `void FLEXIO_SPI_MasterTransferAbort ( FLEXIO_SPI_Type * base, flexio_spi_master_handle_t * handle )`

## Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                     |
| <i>handle</i> | Pointer to the <code>flexio_spi_master_handle_t</code> structure to store the transfer state. |

**23.7.7.25** `status_t FLEXIO_SPI_MasterTransferGetCount ( FLEXIO_SPI_Type * base, flexio_spi_master_handle_t * handle, size_t * count )`

## Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                     |
| <i>handle</i> | Pointer to the <code>flexio_spi_master_handle_t</code> structure to store 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. |

**23.7.7.26** `void FLEXIO_SPI_MasterTransferHandleIRQ ( void * spiType, void * spiHandle )`



## Parameters

|                  |                                                                                               |
|------------------|-----------------------------------------------------------------------------------------------|
| <i>spiType</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                     |
| <i>spiHandle</i> | Pointer to the <code>flexio_spi_master_handle_t</code> structure to store the transfer state. |

**23.7.7.27** `status_t FLEXIO_SPI_SlaveTransferCreateHandle ( FLEXIO_SPI_Type * base, flexio_spi_slave_handle_t * handle, flexio_spi_slave_transfer_callback_t callback, void * userData )`

## Parameters

|                 |                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------|
| <i>base</i>     | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                    |
| <i>handle</i>   | Pointer to the <code>flexio_spi_slave_handle_t</code> structure to store the transfer state. |
| <i>callback</i> | The callback function.                                                                       |
| <i>userData</i> | The parameter of the callback function.                                                      |

## Return values

|                           |                                                |
|---------------------------|------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                |
| <i>kStatus_OutOfRange</i> | The FlexIO type/handle/ISR table out of range. |

**23.7.7.28** `status_t FLEXIO_SPI_SlaveTransferNonBlocking ( FLEXIO_SPI_Type * base, flexio_spi_slave_handle_t * handle, flexio_spi_transfer_t * xfer )`

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

## Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>handle</i> | Pointer to the <code>flexio_spi_slave_handle_t</code> structure to store the transfer state. |
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                    |
| <i>xfer</i>   | FlexIO SPI transfer structure. See <a href="#">flexio_spi_transfer_t</a> .                   |

## Return values

|                                |                                                  |
|--------------------------------|--------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                   |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                       |
| <i>kStatus_FLEXIO_SPI_Busy</i> | SPI is not idle; it is running another transfer. |

**23.7.7.29** `static void FLEXIO_SPI_SlaveTransferAbort ( FLEXIO_SPI_Type * base,  
flexio_spi_slave_handle_t * handle ) [inline], [static]`

## Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                    |
| <i>handle</i> | Pointer to the <code>flexio_spi_slave_handle_t</code> structure to store the transfer state. |

**23.7.7.30** `static status_t FLEXIO_SPI_SlaveTransferGetCount ( FLEXIO_SPI_Type * base, flexio_spi_slave_handle_t * handle, size_t * count ) [inline], [static]`

## Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                    |
| <i>handle</i> | Pointer to the <code>flexio_spi_slave_handle_t</code> structure to store 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. |

**23.7.7.31** `void FLEXIO_SPI_SlaveTransferHandleIRQ ( void * spiType, void * spiHandle )`

## Parameters

|                  |                                                                                              |
|------------------|----------------------------------------------------------------------------------------------|
| <i>spiType</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                    |
| <i>spiHandle</i> | Pointer to the <code>flexio_spi_slave_handle_t</code> structure to store the transfer state. |

## 23.7.8 FlexIO eDMA SPI Driver

### 23.7.8.1 Overview

#### Data Structures

- struct `flexio_spi_master_edma_handle_t`  
*FlexIO SPI eDMA transfer handle, users should not touch the content of the handle. [More...](#)*

#### Typedefs

- typedef  
`flexio_spi_master_edma_handle_t flexio_spi_slave_edma_handle_t`  
*Slave handle is the same with master handle.*
- typedef void(\* `flexio_spi_master_edma_transfer_callback_t`)(`FLEXIO_SPI_Type *base`, `flexio_spi_master_edma_handle_t *handle`, `status_t status`, void \*userData)  
*FlexIO SPI master callback for finished transmit.*
- typedef void(\* `flexio_spi_slave_edma_transfer_callback_t`)(`FLEXIO_SPI_Type *base`, `flexio_spi_slave_edma_handle_t *handle`, `status_t status`, void \*userData)  
*FlexIO SPI slave callback for finished transmit.*

#### Driver version

- #define `FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 0)`)  
*FlexIO SPI EDMA driver version.*

#### eDMA Transactional

- `status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA` (`FLEXIO_SPI_Type *base`, `flexio_spi_master_edma_handle_t *handle`, `flexio_spi_master_edma_transfer_callback_t callback`, void \*userData, `edma_handle_t *txHandle`, `edma_handle_t *rxHandle`)  
*Initializes the FlexIO SPI master eDMA handle.*
- `status_t FLEXIO_SPI_MasterTransferEDMA` (`FLEXIO_SPI_Type *base`, `flexio_spi_master_edma_handle_t *handle`, `flexio_spi_transfer_t *xfer`)  
*Performs a non-blocking FlexIO SPI transfer using eDMA.*
- void `FLEXIO_SPI_MasterTransferAbortEDMA` (`FLEXIO_SPI_Type *base`, `flexio_spi_master_edma_handle_t *handle`)  
*Aborts a FlexIO SPI transfer using eDMA.*
- `status_t FLEXIO_SPI_MasterTransferGetCountEDMA` (`FLEXIO_SPI_Type *base`, `flexio_spi_master_edma_handle_t *handle`, `size_t *count`)  
*Gets the number of bytes transferred so far using FlexIO SPI master eDMA.*
- static void `FLEXIO_SPI_SlaveTransferCreateHandleEDMA` (`FLEXIO_SPI_Type *base`, `flexio_spi_slave_edma_handle_t *handle`, `flexio_spi_slave_edma_transfer_callback_t callback`, void \*userData, `edma_handle_t *txHandle`, `edma_handle_t *rxHandle`)  
*Initializes the FlexIO SPI slave eDMA handle.*

- `status_t FLEXIO_SPI_SlaveTransferEDMA` (`FLEXIO_SPI_Type *base`, `flexio_spi_slave_edma_handle_t *handle`, `flexio_spi_transfer_t *xfer`)  
*Performs a non-blocking FlexIO SPI transfer using eDMA.*
- `static void FLEXIO_SPI_SlaveTransferAbortEDMA` (`FLEXIO_SPI_Type *base`, `flexio_spi_slave_edma_handle_t *handle`)  
*Aborts a FlexIO SPI transfer using eDMA.*
- `static status_t FLEXIO_SPI_SlaveTransferGetCountEDMA` (`FLEXIO_SPI_Type *base`, `flexio_spi_slave_edma_handle_t *handle`, `size_t *count`)  
*Gets the number of bytes transferred so far using FlexIO SPI slave eDMA.*

## 23.7.8.2 Data Structure Documentation

### 23.7.8.2.1 struct flexio\_spi\_master\_edma\_handle

typedef for `flexio_spi_master_edma_handle_t` in advance.

#### Data Fields

- `size_t transferSize`  
*Total bytes to be transferred.*
- `uint8_t nbytes`  
*eDMA minor byte transfer count initially configured.*
- `bool txInProgress`  
*Send transfer in progress.*
- `bool rxInProgress`  
*Receive transfer in progress.*
- `edma_handle_t * txHandle`  
*DMA handler for SPI send.*
- `edma_handle_t * rxHandle`  
*DMA handler for SPI receive.*
- `flexio_spi_master_edma_transfer_callback_t callback`  
*Callback for SPI DMA transfer.*
- `void * userData`  
*User Data for SPI DMA callback.*

## Field Documentation

(1) `size_t flexio_spi_master_edma_handle_t::transferSize`

(2) `uint8_t flexio_spi_master_edma_handle_t::nbytes`

### 23.7.8.3 Macro Definition Documentation

23.7.8.3.1 `#define FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))`

### 23.7.8.4 Typedef Documentation

23.7.8.4.1 `typedef flexio_spi_master_edma_handle_t flexio_spi_slave_edma_handle_t`

### 23.7.8.5 Function Documentation

23.7.8.5.1 `status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA ( FLEXIO_SPI_Type * base, flexio_spi_master_edma_handle_t * handle, flexio_spi_master_edma_transfer_callback_t callback, void * userData, edma_handle_t * txHandle, edma_handle_t * rxHandle )`

This function initializes the FlexIO SPI master eDMA handle which can be used for other FlexIO SPI master transactional APIs. For a specified FlexIO SPI instance, call this API once to get the initialized handle.

#### Parameters

|                 |                                                                                                |
|-----------------|------------------------------------------------------------------------------------------------|
| <i>base</i>     | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                          |
| <i>handle</i>   | Pointer to <code>flexio_spi_master_edma_handle_t</code> structure to store the transfer state. |
| <i>callback</i> | SPI callback, NULL means no callback.                                                          |
| <i>userData</i> | callback function parameter.                                                                   |
| <i>txHandle</i> | User requested eDMA handle for FlexIO SPI RX eDMA transfer.                                    |
| <i>rxHandle</i> | User requested eDMA handle for FlexIO SPI TX eDMA transfer.                                    |

#### Return values

|                           |                                                     |
|---------------------------|-----------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                     |
| <i>kStatus_OutOfRange</i> | The FlexIO SPI eDMA type/handle table out of range. |

23.7.8.5.2 `status_t FLEXIO_SPI_MasterTransferEDMA ( FLEXIO_SPI_Type * base, flexio_spi_master_edma_handle_t * handle, flexio_spi_transfer_t * xfer )`

## Note

This interface returns immediately after transfer initiates. Call `FLEXIO_SPI_MasterGetTransferCountEDMA` to poll the transfer status and check whether the FlexIO SPI transfer is finished.

## Parameters

|               |                                                                                                |
|---------------|------------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                          |
| <i>handle</i> | Pointer to <code>flexio_spi_master_edma_handle_t</code> structure to store the transfer state. |
| <i>xfer</i>   | Pointer to FlexIO SPI transfer structure.                                                      |

## Return values

|                                |                                                      |
|--------------------------------|------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                       |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                           |
| <i>kStatus_FLEXIO_SPI_Busy</i> | FlexIO SPI is not idle, is running another transfer. |

### 23.7.8.5.3 void FLEXIO\_SPI\_MasterTransferAbortEDMA ( FLEXIO\_SPI\_Type \* base, flexio\_spi\_master\_edma\_handle\_t \* handle )

## Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>handle</i> | FlexIO SPI eDMA handle pointer.                       |

### 23.7.8.5.4 status\_t FLEXIO\_SPI\_MasterTransferGetCountEDMA ( FLEXIO\_SPI\_Type \* base, flexio\_spi\_master\_edma\_handle\_t \* handle, size\_t \* count )

## Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure.               |
| <i>handle</i> | FlexIO SPI eDMA handle pointer.                                     |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

**23.7.8.5.5** `static void FLEXIO_SPI_SlaveTransferCreateHandleEDMA ( FLEXIO_SPI_Type * base, flexio_spi_slave_edma_handle_t * handle, flexio_spi_slave_edma_transfer_callback_t callback, void * userData, edma_handle_t * txHandle, edma_handle_t * rxHandle )`  
`[inline], [static]`

This function initializes the FlexIO SPI slave eDMA handle.



## Parameters

|                 |                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>     | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                         |
| <i>handle</i>   | Pointer to <code>flexio_spi_slave_edma_handle_t</code> structure to store the transfer state. |
| <i>callback</i> | SPI callback, NULL means no callback.                                                         |
| <i>userData</i> | callback function parameter.                                                                  |
| <i>txHandle</i> | User requested eDMA handle for FlexIO SPI TX eDMA transfer.                                   |
| <i>rxHandle</i> | User requested eDMA handle for FlexIO SPI RX eDMA transfer.                                   |

### 23.7.8.5.6 `status_t FLEXIO_SPI_SlaveTransferEDMA ( FLEXIO_SPI_Type * base, flexio_spi_slave_edma_handle_t * handle, flexio_spi_transfer_t * xfer )`

## Note

This interface returns immediately after transfer initiates. Call `FLEXIO_SPI_SlaveGetTransferCountEDMA` to poll the transfer status and check whether the FlexIO SPI transfer is finished.

## Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                         |
| <i>handle</i> | Pointer to <code>flexio_spi_slave_edma_handle_t</code> structure to store the transfer state. |
| <i>xfer</i>   | Pointer to FlexIO SPI transfer structure.                                                     |

## Return values

|                                |                                                      |
|--------------------------------|------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                       |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                           |
| <i>kStatus_FLEXIO_SPI_Busy</i> | FlexIO SPI is not idle, is running another transfer. |

### 23.7.8.5.7 `static void FLEXIO_SPI_SlaveTransferAbortEDMA ( FLEXIO_SPI_Type * base, flexio_spi_slave_edma_handle_t * handle ) [inline], [static]`

## Parameters

|               |                                                                                  |
|---------------|----------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                            |
| <i>handle</i> | Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state. |

**23.7.8.5.8** static status\_t FLEXIO\_SPI\_SlaveTransferGetCountEDMA ( FLEXIO\_SPI\_Type \* *base*, flexio\_spi\_slave\_edma\_handle\_t \* *handle*, size\_t \* *count* ) [inline], [static]

Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure.               |
| <i>handle</i> | FlexIO SPI eDMA handle pointer.                                     |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

## 23.8 FlexIO UART Driver

### 23.8.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Universal Asynchronous Receiver/Transmitter (UART) function using the Flexible I/O.

FlexIO UART driver includes functional APIs and transactional APIs. Functional APIs target low-level APIs. Functional APIs can be used for the FlexIO UART initialization/configuration/operation for optimization/customization purpose. Using the functional APIs requires the knowledge of the FlexIO UART peripheral and how to organize functional APIs to meet the application requirements. All functional API use the `FLEXIO_UART_Type *` as the first parameter. FlexIO UART functional operation groups provide the functional APIs set.

Transactional APIs target high-level APIs. Transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the `flexio_uart_handle_t` as the second parameter. Initialize the handle by calling the `FLEXIO_UART_TransferCreateHandle()` API.

Transactional APIs support asynchronous transfer. This means that the functions `FLEXIO_UART_SendNonBlocking()` and `FLEXIO_UART_ReceiveNonBlocking()` set up an interrupt for data transfer. When the transfer is complete, the upper layer is notified through a callback function with the `kStatus_FLEXIO_UART_TxIdle` and `kStatus_FLEXIO_UART_RxIdle` status.

Transactional receive APIs support the ring buffer. Prepare the memory for the ring buffer and pass in the start address and size through calling the `FLEXIO_UART_InstallRingBuffer()`. When the ring buffer is enabled, the received data is saved to the ring buffer in the background. The function `FLEXIO_UART_ReceiveNonBlocking()` first gets data from the ring buffer. If ring buffer does not have enough data, the function returns the data to the ring buffer and saves the received data to user memory. When all data is received, the upper layer is informed through a callback with the `statuskStatus_FLEXIO_UART_RxIdle` status.

If the receive ring buffer is full, the upper layer is informed through a callback with status `kStatus_FLEXIO_UART_RxRingBufferOverrun`. In the callback function, the upper layer reads data from the ring buffer. If not, the oldest data is overwritten by the new data.

The ring buffer size is specified when calling the `FLEXIO_UART_InstallRingBuffer`. Note that one byte is reserved for the ring buffer maintenance. Create a handle as follows.

```
FLEXIO_UART_InstallRingBuffer(&uartDev, &handle, &ringBuffer, 32);
```

In this example, the buffer size is 32. However, only 31 bytes are used for saving data.

### 23.8.2 Typical use case

#### 23.8.2.1 FlexIO UART send/receive using a polling method

```
uint8_t ch;
```

```

FLEXIO_UART_Type uartDev;
status_t result = kStatus_Success;
flexio_uart_user_config user_config;
FLEXIO_UART_GetDefaultConfig(&user_config);
user_config.baudRate_Bps = 115200U;
user_config.enableUart = true;

uartDev.flexioBase = BOARD_FLEXIO_BASE;
uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
uartDev.shifterIndex[0] = 0U;
uartDev.shifterIndex[1] = 1U;
uartDev.timerIndex[0] = 0U;
uartDev.timerIndex[1] = 1U;

result = FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);
//Check if configuration is correct.
if(result != kStatus_Success)
{
 return;
}
FLEXIO_UART_WriteBlocking(&uartDev, txbuff, sizeof(txbuff));

while(1)
{
 FLEXIO_UART_ReadBlocking(&uartDev, &ch, 1);
 FLEXIO_UART_WriteBlocking(&uartDev, &ch, 1);
}

```

### 23.8.2.2 FlexIO UART send/receive using an interrupt method

```

FLEXIO_UART_Type uartDev;
flexio_uart_handle_t g_uartHandle;
flexio_uart_config_t user_config;
flexio_uart_transfer_t sendXfer;
flexio_uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = {'H', 'e', 'l', 'l', 'o'};
uint8_t receiveData[32];

void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
 status_t status, void *userData)
{
 userData = userData;

 if (kStatus_FLEXIO_UART_TxIdle == status)
 {
 txFinished = true;
 }

 if (kStatus_FLEXIO_UART_RxIdle == status)
 {
 rxFinished = true;
 }
}

void main(void)
{
 //...

 FLEXIO_UART_GetDefaultConfig(&user_config);
 user_config.baudRate_Bps = 115200U;
 user_config.enableUart = true;

```

```

uartDev.flexioBase = BOARD_FLEXIO_BASE;
uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
uartDev.shifterIndex[0] = 0U;
uartDev.shifterIndex[1] = 1U;
uartDev.timerIndex[0] = 0U;
uartDev.timerIndex[1] = 1U;

result = FLEXIO_UART_Init(&uartDev, &user_config, 120000000U);
//Check if configuration is correct.
if(result != kStatus_Success)
{
 return;
}

FLEXIO_UART_TransferCreateHandle(&uartDev, &g_uartHandle,
 FLEXIO_UART_UserCallback, NULL);

// Prepares to send.
sendXfer.data = sendData;
sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
txFinished = false;

// Sends out.
FLEXIO_UART_SendNonBlocking(&uartDev, &g_uartHandle, &sendXfer);

// Send finished.
while (!txFinished)
{
}

// Prepares to receive.
receiveXfer.data = receiveData;
receiveXfer.dataSize = sizeof(receiveData)/sizeof(receiveData[0]);
rxFinished = false;

// Receives.
FLEXIO_UART_ReceiveNonBlocking(&uartDev, &g_uartHandle, &receiveXfer, NULL);

// Receive finished.
while (!rxFinished)
{
}

// ...
}

```

### 23.8.2.3 FlexIO UART receive using the ringbuffer feature

```

#define RING_BUFFER_SIZE 64
#define RX_DATA_SIZE 32

FLEXIO_UART_Type uartDev;
flexio_uart_handle_t g_uartHandle;
flexio_uart_config_t user_config;
flexio_uart_transfer_t sendXfer;
flexio_uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t receiveData[RX_DATA_SIZE];
uint8_t ringBuffer[RING_BUFFER_SIZE];

void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
 status_t status, void *userData)
{

```

```

 userData = userData;

 if (kStatus_FLEXIO_UART_RxIdle == status)
 {
 rxFinished = true;
 }
}

void main(void)
{
 size_t bytesRead;
 //...

 FLEXIO_UART_GetDefaultConfig(&user_config);
 user_config.baudRate_Bps = 115200U;
 user_config.enableUart = true;

 uartDev.flexioBase = BOARD_FLEXIO_BASE;
 uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
 uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
 uartDev.shifterIndex[0] = 0U;
 uartDev.shifterIndex[1] = 1U;
 uartDev.timerIndex[0] = 0U;
 uartDev.timerIndex[1] = 1U;

 result = FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);
 //Check if configuration is correct.
 if(result != kStatus_Success)
 {
 return;
 }

 FLEXIO_UART_TransferCreateHandle(&uartDev, &g_uartHandle,
 FLEXIO_UART_UserCallback, NULL);
 FLEXIO_UART_InstallRingBuffer(&uartDev, &g_uartHandle, ringBuffer, RING_BUFFER_SIZE);

 // Receive is working in the background to the ring buffer.

 // Prepares to receive.
 receiveXfer.data = receiveData;
 receiveXfer.dataSize = RX_DATA_SIZE;
 rxFinished = false;

 // Receives.
 FLEXIO_UART_ReceiveNonBlocking(&uartDev, &g_uartHandle, &receiveXfer, &bytesRead);

 if (bytesRead == RX_DATA_SIZE) /* Have read enough data. */
 {
 ;
 }
 else
 {
 if (bytesRead) /* Received some data, process first. */
 {
 ;
 }

 // Receive finished.
 while (!rxFinished)
 {
 }
 }

 // ...
}

```

### 23.8.2.4 FlexIO UART send/receive using a DMA method

```

FLEXIO_UART_Type uartDev;
flexio_uart_handle_t g_uartHandle;
dma_handle_t g_uartTxDmaHandle;
dma_handle_t g_uartRxDmaHandle;
flexio_uart_config_t user_config;
flexio_uart_transfer_t sendXfer;
flexio_uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = {'H', 'e', 'l', 'l', 'o'};
uint8_t receiveData[32];

void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
 status_t status, void *userData)
{
 userData = userData;

 if (kStatus_FLEXIO_UART_TxIdle == status)
 {
 txFinished = true;
 }

 if (kStatus_FLEXIO_UART_RxIdle == status)
 {
 rxFinished = true;
 }
}

void main(void)
{
 //...

 FLEXIO_UART_GetDefaultConfig(&user_config);
 user_config.baudRate_Bps = 115200U;
 user_config.enableUart = true;

 uartDev.flexioBase = BOARD_FLEXIO_BASE;
 uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
 uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
 uartDev.shifterIndex[0] = 0U;
 uartDev.shifterIndex[1] = 1U;
 uartDev.timerIndex[0] = 0U;
 uartDev.timerIndex[1] = 1U;
 result = FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);
 //Check if configuration is correct.
 if(result != kStatus_Success)
 {
 return;
 }

 /* Init DMAMUX. */
 DMAMUX_Init(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR)

 /* Init the DMA/EDMA module */
 #if defined(FSL_FEATURE_SOC_DMA_COUNT) && FSL_FEATURE_SOC_DMA_COUNT > 0U
 DMA_Init(EXAMPLE_FLEXIO_UART_DMA_BASEADDR);
 DMA_CreateHandle(&g_uartTxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR, FLEXIO_UART_TX_DMA_CHANNEL);
 DMA_CreateHandle(&g_uartRxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR, FLEXIO_UART_RX_DMA_CHANNEL);
 #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_FLEXIO_UART_DMA_BASEADDR, &edmaConfig);
 #endif
}

```

```

 EDMA_CreateHandle(&g_uartTxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR,
 FLEXIO_UART_TX_DMA_CHANNEL);
 EDMA_CreateHandle(&g_uartRxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR,
 FLEXIO_UART_RX_DMA_CHANNEL);
#endif /* FSL_FEATURE_SOC_EDMA_COUNT */

 dma_request_source_tx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + uartDev.
 shifterIndex[0]);
 dma_request_source_rx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + uartDev.
 shifterIndex[1]);

 /* Requests DMA channels for transmit and receive. */
 DMAMUX_SetSource(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR, FLEXIO_UART_TX_DMA_CHANNEL, (
 dma_request_source_t)dma_request_source_tx);
 DMAMUX_SetSource(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR, FLEXIO_UART_RX_DMA_CHANNEL, (
 dma_request_source_t)dma_request_source_rx);
 DMAMUX_EnableChannel(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR,
 FLEXIO_UART_TX_DMA_CHANNEL);
 DMAMUX_EnableChannel(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR,
 FLEXIO_UART_RX_DMA_CHANNEL);

 FLEXIO_UART_TransferCreateHandleDMA(&uartDev, &g_uartHandle, FLEXIO_UART_UserCallback, NULL, &
 g_uartTxDmaHandle, &g_uartRxDmaHandle);

 // Prepares to send.
 sendXfer.data = sendData
 sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
 txFinished = false;

 // Sends out.
 FLEXIO_UART_SendDMA(&uartDev, &g_uartHandle, &sendXfer);

 // Send finished.
 while (!txFinished)
 {
 }

 // Prepares to receive.
 receiveXfer.data = receiveData;
 receiveXfer.dataSize = sizeof(receiveData)/sizeof(receiveData[0]);
 rxFinished = false;

 // Receives.
 FLEXIO_UART_ReceiveDMA(&uartDev, &g_uartHandle, &receiveXfer, NULL);

 // Receive finished.
 while (!rxFinished)
 {
 }

 // ...
}

```

## Modules

- [FlexIO eDMA UART Driver](#)

## Data Structures

- struct [FLEXIO\\_UART\\_Type](#)  
Define FlexIO UART access structure typedef. [More...](#)



- struct `flexio_uart_config_t`  
Define FlexIO UART user configuration structure. [More...](#)
- struct `flexio_uart_transfer_t`  
Define FlexIO UART transfer structure. [More...](#)
- struct `flexio_uart_handle_t`  
Define FLEXIO 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(\* `flexio_uart_transfer_callback_t`)(FLEXIO\_UART\_Type \*base, flexio\_uart\_handle\_t \*handle, `status_t` status, void \*userData)  
*FlexIO UART transfer callback function.*

## Enumerations

- enum {  
`kStatus_FLEXIO_UART_TxBusy` = MAKE\_STATUS(kStatusGroup\_FLEXIO\_UART, 0),  
`kStatus_FLEXIO_UART_RxBusy` = MAKE\_STATUS(kStatusGroup\_FLEXIO\_UART, 1),  
`kStatus_FLEXIO_UART_TxIdle` = MAKE\_STATUS(kStatusGroup\_FLEXIO\_UART, 2),  
`kStatus_FLEXIO_UART_RxIdle` = MAKE\_STATUS(kStatusGroup\_FLEXIO\_UART, 3),  
`kStatus_FLEXIO_UART_ERROR` = MAKE\_STATUS(kStatusGroup\_FLEXIO\_UART, 4),  
`kStatus_FLEXIO_UART_RxRingBufferOverrun`,  
`kStatus_FLEXIO_UART_RxHardwareOverrun` = MAKE\_STATUS(kStatusGroup\_FLEXIO\_UART, 6),  
`kStatus_FLEXIO_UART_Timeout` = MAKE\_STATUS(kStatusGroup\_FLEXIO\_UART, 7),  
`kStatus_FLEXIO_UART_BaudrateNotSupport` }  
*Error codes for the UART driver.*
- enum `flexio_uart_bit_count_per_char_t` {  
`kFLEXIO_UART_7BitsPerChar` = 7U,  
`kFLEXIO_UART_8BitsPerChar` = 8U,  
`kFLEXIO_UART_9BitsPerChar` = 9U }  
*FlexIO UART bit count per char.*
- enum `_flexio_uart_interrupt_enable` {  
`kFLEXIO_UART_TxDataRegEmptyInterruptEnable` = 0x1U,  
`kFLEXIO_UART_RxDataRegFullInterruptEnable` = 0x2U }  
*Define FlexIO UART interrupt mask.*
- enum `_flexio_uart_status_flags` {

```

kFLEXIO_UART_TxDataRegEmptyFlag = 0x1U,
kFLEXIO_UART_RxDataRegFullFlag = 0x2U,
kFLEXIO_UART_RxOverRunFlag = 0x4U }

```

*Define FlexIO UART status mask.*

## Driver version

- #define `FSL_FLEXIO_UART_DRIVER_VERSION` (`MAKE_VERSION(2, 4, 0)`)  
*FlexIO UART driver version.*

## Initialization and deinitialization

- `status_t FLEXIO_UART_Init` (`FLEXIO_UART_Type *base`, `const flexio_uart_config_t *userConfig`, `uint32_t srcClock_Hz`)  
*Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration.*
- `void FLEXIO_UART_Deinit` (`FLEXIO_UART_Type *base`)  
*Resets the FlexIO UART shifter and timer config.*
- `void FLEXIO_UART_GetDefaultConfig` (`flexio_uart_config_t *userConfig`)  
*Gets the default configuration to configure the FlexIO UART.*

## Status

- `uint32_t FLEXIO_UART_GetStatusFlags` (`FLEXIO_UART_Type *base`)  
*Gets the FlexIO UART status flags.*
- `void FLEXIO_UART_ClearStatusFlags` (`FLEXIO_UART_Type *base`, `uint32_t mask`)  
*Gets the FlexIO UART status flags.*

## Interrupts

- `void FLEXIO_UART_EnableInterrupts` (`FLEXIO_UART_Type *base`, `uint32_t mask`)  
*Enables the FlexIO UART interrupt.*
- `void FLEXIO_UART_DisableInterrupts` (`FLEXIO_UART_Type *base`, `uint32_t mask`)  
*Disables the FlexIO UART interrupt.*

## DMA Control

- `static uint32_t FLEXIO_UART_GetTxDataRegisterAddress` (`FLEXIO_UART_Type *base`)  
*Gets the FlexIO UART transmit data register address.*
- `static uint32_t FLEXIO_UART_GetRxDataRegisterAddress` (`FLEXIO_UART_Type *base`)  
*Gets the FlexIO UART receive data register address.*
- `static void FLEXIO_UART_EnableTxDMA` (`FLEXIO_UART_Type *base`, `bool enable`)  
*Enables/disables the FlexIO UART transmit DMA.*
- `static void FLEXIO_UART_EnableRxDMA` (`FLEXIO_UART_Type *base`, `bool enable`)

*Enables/disables the FlexIO UART receive DMA.*

## Bus Operations

- static void `FLEXIO_UART_Enable` (`FLEXIO_UART_Type *base`, bool enable)  
*Enables/disables the FlexIO UART module operation.*
- static void `FLEXIO_UART_WriteByte` (`FLEXIO_UART_Type *base`, const uint8\_t \*buffer)  
*Writes one byte of data.*
- static void `FLEXIO_UART_ReadByte` (`FLEXIO_UART_Type *base`, uint8\_t \*buffer)  
*Reads one byte of data.*
- `status_t FLEXIO_UART_WriteBlocking` (`FLEXIO_UART_Type *base`, const uint8\_t \*txData, size\_t txSize)  
*Sends a buffer of data bytes.*
- `status_t FLEXIO_UART_ReadBlocking` (`FLEXIO_UART_Type *base`, uint8\_t \*rxData, size\_t rxSize)  
*Receives a buffer of bytes.*

## Transactional

- `status_t FLEXIO_UART_TransferCreateHandle` (`FLEXIO_UART_Type *base`, flexio\_uart\_handle\_t \*handle, flexio\_uart\_transfer\_callback\_t callback, void \*userData)  
*Initializes the UART handle.*
- void `FLEXIO_UART_TransferStartRingBuffer` (`FLEXIO_UART_Type *base`, flexio\_uart\_handle\_t \*handle, uint8\_t \*ringBuffer, size\_t ringBufferSize)  
*Sets up the RX ring buffer.*
- void `FLEXIO_UART_TransferStopRingBuffer` (`FLEXIO_UART_Type *base`, flexio\_uart\_handle\_t \*handle)  
*Aborts the background transfer and uninstalls the ring buffer.*
- `status_t FLEXIO_UART_TransferSendNonBlocking` (`FLEXIO_UART_Type *base`, flexio\_uart\_handle\_t \*handle, flexio\_uart\_transfer\_t \*xfer)  
*Transmits a buffer of data using the interrupt method.*
- void `FLEXIO_UART_TransferAbortSend` (`FLEXIO_UART_Type *base`, flexio\_uart\_handle\_t \*handle)  
*Aborts the interrupt-driven data transmit.*
- `status_t FLEXIO_UART_TransferGetSendCount` (`FLEXIO_UART_Type *base`, flexio\_uart\_handle\_t \*handle, size\_t \*count)  
*Gets the number of bytes sent.*
- `status_t FLEXIO_UART_TransferReceiveNonBlocking` (`FLEXIO_UART_Type *base`, flexio\_uart\_handle\_t \*handle, flexio\_uart\_transfer\_t \*xfer, size\_t \*receivedBytes)  
*Receives a buffer of data using the interrupt method.*
- void `FLEXIO_UART_TransferAbortReceive` (`FLEXIO_UART_Type *base`, flexio\_uart\_handle\_t \*handle)  
*Aborts the receive data which was using IRQ.*
- `status_t FLEXIO_UART_TransferGetReceiveCount` (`FLEXIO_UART_Type *base`, flexio\_uart\_handle\_t \*handle, size\_t \*count)  
*Gets the number of bytes received.*
- void `FLEXIO_UART_TransferHandleIRQ` (void \*uartType, void \*uartHandle)

*FlexIO UART IRQ handler function.*

### 23.8.3 Data Structure Documentation

#### 23.8.3.1 struct FLEXIO\_UART\_Type

##### Data Fields

- FLEXIO\_Type \* [flexioBase](#)  
*FlexIO base pointer.*
- uint8\_t [TxPinIndex](#)  
*Pin select for UART\_Tx.*
- uint8\_t [RxPinIndex](#)  
*Pin select for UART\_Rx.*
- uint8\_t [shifterIndex](#) [2]  
*Shifter index used in FlexIO UART.*
- uint8\_t [timerIndex](#) [2]  
*Timer index used in FlexIO UART.*

##### Field Documentation

- (1) FLEXIO\_Type\* FLEXIO\_UART\_Type::flexioBase
- (2) uint8\_t FLEXIO\_UART\_Type::TxPinIndex
- (3) uint8\_t FLEXIO\_UART\_Type::RxPinIndex
- (4) uint8\_t FLEXIO\_UART\_Type::shifterIndex[2]
- (5) uint8\_t FLEXIO\_UART\_Type::timerIndex[2]

#### 23.8.3.2 struct flexio\_uart\_config\_t

##### Data Fields

- bool [enableUart](#)  
*Enable/disable FlexIO UART TX & RX.*
- bool [enableInDoze](#)  
*Enable/disable FlexIO operation in doze mode.*
- bool [enableInDebug](#)  
*Enable/disable FlexIO operation in debug mode.*
- bool [enableFastAccess](#)  
*Enable/disable fast access to FlexIO registers,  
fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- uint32\_t [baudRate\\_Bps](#)  
*Baud rate in Bps.*
- [flexio\\_uart\\_bit\\_count\\_per\\_char\\_t](#) [bitCountPerChar](#)  
*number of bits, 7/8/9 -bit*

**Field Documentation**

- (1) `bool flexio_uart_config_t::enableUart`
- (2) `bool flexio_uart_config_t::enableFastAccess`
- (3) `uint32_t flexio_uart_config_t::baudRate_Bps`

**23.8.3.3 struct flexio\_uart\_transfer\_t****Data Fields**

- `size_t dataSize`  
*Transfer size.*
- `uint8_t * data`  
*The buffer of data to be transfer.*
- `uint8_t * rxData`  
*The buffer to receive data.*
- `const uint8_t * txData`  
*The buffer of data to be sent.*

**Field Documentation**

- (1) `uint8_t* flexio_uart_transfer_t::data`
- (2) `uint8_t* flexio_uart_transfer_t::rxData`
- (3) `const uint8_t* flexio_uart_transfer_t::txData`

**23.8.3.4 struct \_flexio\_uart\_handle****Data Fields**

- `const uint8_t *volatile txData`  
*Address of remaining data to send.*
- `volatile size_t txDataSize`  
*Size of the remaining data to send.*
- `uint8_t *volatile rxData`  
*Address of remaining data to receive.*
- `volatile size_t rxDataSize`  
*Size of the remaining data to receive.*
- `size_t txDataSizeAll`  
*Total bytes to be sent.*
- `size_t rxDataSizeAll`  
*Total bytes to be received.*
- `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.*
- `flexio_uart_transfer_callback_t` `callback`  
*Callback function.*
- void \* `userData`  
*UART callback function parameter.*
- volatile uint8\_t `txState`  
*TX transfer state.*
- volatile uint8\_t `rxState`  
*RX transfer state.*

## Field Documentation

- (1) `const uint8_t* volatile flexio_uart_handle_t::txData`
- (2) `volatile size_t flexio_uart_handle_t::txDataSize`
- (3) `uint8_t* volatile flexio_uart_handle_t::rxData`
- (4) `volatile size_t flexio_uart_handle_t::rxDataSize`
- (5) `size_t flexio_uart_handle_t::txDataSizeAll`
- (6) `size_t flexio_uart_handle_t::rxDataSizeAll`
- (7) `uint8_t* flexio_uart_handle_t::rxRingBuffer`
- (8) `size_t flexio_uart_handle_t::rxRingBufferSize`
- (9) `volatile uint16_t flexio_uart_handle_t::rxRingBufferHead`
- (10) `volatile uint16_t flexio_uart_handle_t::rxRingBufferTail`
- (11) `flexio_uart_transfer_callback_t flexio_uart_handle_t::callback`
- (12) `void* flexio_uart_handle_t::userData`
- (13) `volatile uint8_t flexio_uart_handle_t::txState`

## 23.8.4 Macro Definition Documentation

23.8.4.1 `#define FSL_FLEXIO_UART_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))`

23.8.4.2 `#define UART_RETRY_TIMES 0U /* Defining to zero means to keep waiting for the flag until it is assert/deassert. */`

## 23.8.5 Typedef Documentation

23.8.5.1 `typedef void(* flexio_uart_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, status_t status, void *userData)`

## 23.8.6 Enumeration Type Documentation

### 23.8.6.1 anonymous enum

Enumerator

*kStatus\_FLEXIO\_UART\_TxBusy* Transmitter is busy.

*kStatus\_FLEXIO\_UART\_RxBusy* Receiver is busy.

*kStatus\_FLEXIO\_UART\_TxIdle* UART transmitter is idle.  
*kStatus\_FLEXIO\_UART\_RxIdle* UART receiver is idle.  
*kStatus\_FLEXIO\_UART\_ERROR* ERROR happens on UART.  
*kStatus\_FLEXIO\_UART\_RxRingBufferOverrun* UART RX software ring buffer overrun.  
*kStatus\_FLEXIO\_UART\_RxHardwareOverrun* UART RX receiver overrun.  
*kStatus\_FLEXIO\_UART\_Timeout* UART times out.  
*kStatus\_FLEXIO\_UART\_BaudrateNotSupport* Baudrate is not supported in current clock source.

### 23.8.6.2 enum flexio\_uart\_bit\_count\_per\_char\_t

Enumerator

*kFLEXIO\_UART\_7BitsPerChar* 7-bit data characters  
*kFLEXIO\_UART\_8BitsPerChar* 8-bit data characters  
*kFLEXIO\_UART\_9BitsPerChar* 9-bit data characters

### 23.8.6.3 enum \_flexio\_uart\_interrupt\_enable

Enumerator

*kFLEXIO\_UART\_TxDataRegEmptyInterruptEnable* Transmit buffer empty interrupt enable.  
*kFLEXIO\_UART\_RxDataRegFullInterruptEnable* Receive buffer full interrupt enable.

### 23.8.6.4 enum \_flexio\_uart\_status\_flags

Enumerator

*kFLEXIO\_UART\_TxDataRegEmptyFlag* Transmit buffer empty flag.  
*kFLEXIO\_UART\_RxDataRegFullFlag* Receive buffer full flag.  
*kFLEXIO\_UART\_RxOverRunFlag* Receive buffer over run flag.

## 23.8.7 Function Documentation

### 23.8.7.1 status\_t FLEXIO\_UART\_Init ( FLEXIO\_UART\_Type \* base, const flexio\_uart\_config\_t \* userConfig, uint32\_t srcClock\_Hz )

The configuration structure can be filled by the user or be set with default values by [FLEXIO\\_UART\\_GetDefaultConfig\(\)](#).

Example



```

FLEXIO_UART_Type base = {
 .flexioBase = FLEXIO,
 .TxPinIndex = 0,
 .RxPinIndex = 1,
 .shifterIndex = {0,1},
 .timerIndex = {0,1}
};
flexio_uart_config_t config = {
 .enableInDoze = false,
 .enableInDebug = true,
 .enableFastAccess = false,
 .baudRate_Bps = 115200U,
 .bitCountPerChar = 8
};
FLEXIO_UART_Init(base, &config, srcClock_Hz);

```

### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>base</i>        | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.     |
| <i>userConfig</i>  | Pointer to the <a href="#">flexio_uart_config_t</a> structure. |
| <i>srcClock_Hz</i> | FlexIO source clock in Hz.                                     |

### Return values

|                                                     |                                                               |
|-----------------------------------------------------|---------------------------------------------------------------|
| <i>kStatus_Success</i>                              | Configuration success.                                        |
| <i>kStatus_FLEXIO_UART-<br/>_BaudrateNotSupport</i> | Baudrate is not supported for current clock source frequency. |

### 23.8.7.2 void FLEXIO\_UART\_Deinit ( FLEXIO\_UART\_Type \* base )

#### Note

After calling this API, call the `FLEXIO_UART_Init` to use the FlexIO UART module.

### Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_UART_Type</a> structure |
|-------------|-------------------------------------------------------|

### 23.8.7.3 void FLEXIO\_UART\_GetDefaultConfig ( flexio\_uart\_config\_t \* userConfig )

The configuration can be used directly for calling the `FLEXIO_UART_Init()`. Example:

```

flexio_uart_config_t config;
FLEXIO_UART_GetDefaultConfig(&userConfig);

```

Parameters

|                   |                                                                |
|-------------------|----------------------------------------------------------------|
| <i>userConfig</i> | Pointer to the <a href="#">flexio_uart_config_t</a> structure. |
|-------------------|----------------------------------------------------------------|

#### 23.8.7.4 uint32\_t FLEXIO\_UART\_GetStatusFlags ( FLEXIO\_UART\_Type \* *base* )

Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
|-------------|------------------------------------------------------------|

Returns

FlexIO UART status flags.

#### 23.8.7.5 void FLEXIO\_UART\_ClearStatusFlags ( FLEXIO\_UART\_Type \* *base*, uint32\_t *mask* )

Parameters

|             |                                                                                                                                                                                                                                          |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.                                                                                                                                                                               |
| <i>mask</i> | Status flag. The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• kFLEXIO_UART_TxDataRegEmptyFlag</li> <li>• kFLEXIO_UART_RxEmptyFlag</li> <li>• kFLEXIO_UART_RxOverRunFlag</li> </ul> |

#### 23.8.7.6 void FLEXIO\_UART\_EnableInterrupts ( FLEXIO\_UART\_Type \* *base*, uint32\_t *mask* )

This function enables the FlexIO UART interrupt.

Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
| <i>mask</i> | Interrupt source.                                          |

#### 23.8.7.7 void FLEXIO\_UART\_DisableInterrupts ( FLEXIO\_UART\_Type \* *base*, uint32\_t *mask* )

This function disables the FlexIO UART interrupt.

Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
| <i>mask</i> | Interrupt source.                                          |

#### 23.8.7.8 **static uint32\_t FLEXIO\_UART\_GetTxDataRegisterAddress ( FLEXIO\_UART\_Type \* *base* ) [inline], [static]**

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
|-------------|------------------------------------------------------------|

Returns

FlexIO UART transmit data register address.

#### 23.8.7.9 **static uint32\_t FLEXIO\_UART\_GetRxDataRegisterAddress ( FLEXIO\_UART\_Type \* *base* ) [inline], [static]**

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
|-------------|------------------------------------------------------------|

Returns

FlexIO UART receive data register address.

#### 23.8.7.10 **static void FLEXIO\_UART\_EnableTxDMA ( FLEXIO\_UART\_Type \* *base*, bool *enable* ) [inline], [static]**

This function enables/disables the FlexIO UART Tx DMA, which means asserting the kFLEXIO\_UART\_TxDataRegEmptyFlag does/doesn't trigger the DMA request.

## Parameters

|               |                                                            |
|---------------|------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
| <i>enable</i> | True to enable, false to disable.                          |

### 23.8.7.11 static void FLEXIO\_UART\_EnableRxDMA ( FLEXIO\_UART\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables/disables the FlexIO UART Rx DMA, which means asserting kFLEXIO\_UART\_RxDataRegFullFlag does/doesn't trigger the DMA request.

## Parameters

|               |                                                            |
|---------------|------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
| <i>enable</i> | True to enable, false to disable.                          |

### 23.8.7.12 static void FLEXIO\_UART\_Enable ( FLEXIO\_UART\_Type \* *base*, bool *enable* ) [inline], [static]

## Parameters

|               |                                                   |
|---------------|---------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> . |
| <i>enable</i> | True to enable, false does not have any effect.   |

### 23.8.7.13 static void FLEXIO\_UART\_WriteByte ( FLEXIO\_UART\_Type \* *base*, const uint8\_t \* *buffer* ) [inline], [static]

## Note

This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the TxEmptyFlag is asserted before calling this API.

## Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
|-------------|------------------------------------------------------------|

|               |                         |
|---------------|-------------------------|
| <i>buffer</i> | The data bytes to send. |
|---------------|-------------------------|

**23.8.7.14** `static void FLEXIO_UART_ReadByte ( FLEXIO_UART_Type * base, uint8_t * buffer ) [inline], [static]`

Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

|               |                                                            |
|---------------|------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
| <i>buffer</i> | The buffer to store the received bytes.                    |

**23.8.7.15** `status_t FLEXIO_UART_WriteBlocking ( FLEXIO_UART_Type * base, const uint8_t * txData, size_t txSize )`

Note

This function blocks using the polling method until all bytes have been sent.

Parameters

|               |                                                            |
|---------------|------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
| <i>txData</i> | The data bytes to send.                                    |
| <i>txSize</i> | The number of data bytes to send.                          |

Return values

|                                          |                                         |
|------------------------------------------|-----------------------------------------|
| <i>kStatus_FLEXIO_UART-<br/>_Timeout</i> | Transmission timed out and was aborted. |
| <i>kStatus_Success</i>                   | Successfully wrote all data.            |

**23.8.7.16** `status_t FLEXIO_UART_ReadBlocking ( FLEXIO_UART_Type * base, uint8_t * rxData, size_t rxSize )`

Note

This function blocks using the polling method until all bytes have been received.

## Parameters

|               |                                                            |
|---------------|------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
| <i>rxData</i> | The buffer to store the received bytes.                    |
| <i>rxSize</i> | The number of data bytes to be received.                   |

## Return values

|                                          |                                         |
|------------------------------------------|-----------------------------------------|
| <i>kStatus_FLEXIO_UART-<br/>_Timeout</i> | Transmission timed out and was aborted. |
| <i>kStatus_Success</i>                   | Successfully received all data.         |

**23.8.7.17** `status_t FLEXIO_UART_TransferCreateHandle ( FLEXIO_UART_Type * base, flexio_uart_handle_t * handle, flexio_uart_transfer_callback_t callback, void * userData )`

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the "background" receiving, which means that users can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the [FLEXIO\\_UART\\_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as `ringBuffer`.

## Parameters

|                 |                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------|
| <i>base</i>     | to <a href="#">FLEXIO_UART_Type</a> structure.                                          |
| <i>handle</i>   | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state. |
| <i>callback</i> | The callback function.                                                                  |
| <i>userData</i> | The parameter of the callback function.                                                 |

## Return values

|                           |                                                |
|---------------------------|------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                |
| <i>kStatus_OutOfRange</i> | The FlexIO type/handle/ISR table out of range. |

**23.8.7.18** `void FLEXIO_UART_TransferStartRingBuffer ( FLEXIO_UART_Type * base, flexio_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 is stored into the ring buffer even when the user doesn't

call the `UART_ReceiveNonBlocking()` API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly.

#### Note

When using the RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, only 31 bytes are used for saving data.

#### Parameters

|                       |                                                                                              |
|-----------------------|----------------------------------------------------------------------------------------------|
| <i>base</i>           | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.                                   |
| <i>handle</i>         | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state.      |
| <i>ringBuffer</i>     | Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer. |
| <i>ringBufferSize</i> | Size of the ring buffer.                                                                     |

#### 23.8.7.19 void FLEXIO\_UART\_TransferStopRingBuffer ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_handle\_t \* *handle* )

This function aborts the background transfer and uninstalls the ring buffer.

#### Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state. |

#### 23.8.7.20 status\_t FLEXIO\_UART\_TransferSendNonBlocking ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_handle\_t \* *handle*, flexio\_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 ISR, the FlexIO UART driver calls the callback function and passes the [kStatus\\_FLEXIO\\_UART\\_TxIdle](#) as status parameter.

#### Note

The `kStatus_FLEXIO_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.

## Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state. |
| <i>xfer</i>   | FlexIO UART transfer structure. See <a href="#">flexio_uart_transfer_t</a> .            |

## Return values

|                            |                                                                                |
|----------------------------|--------------------------------------------------------------------------------|
| <i>kStatus_Success</i>     | Successfully starts the data transmission.                                     |
| <i>kStatus_UART_TxBusy</i> | Previous transmission still not finished, data not written to the TX register. |

### 23.8.7.21 void FLEXIO\_UART\_TransferAbortSend ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_handle\_t \* *handle* )

This function aborts the interrupt-driven data sending. Get the `remainBytes` to find out how many bytes are still not sent out.

## Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state. |

### 23.8.7.22 status\_t FLEXIO\_UART\_TransferGetSendCount ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_handle\_t \* *handle*, size\_t \* *count* )

This function gets the number of bytes sent driven by interrupt.

## Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state. |
| <i>count</i>  | Number of bytes sent so far by the non-blocking transaction.                            |

## Return values

|                                      |                                                   |
|--------------------------------------|---------------------------------------------------|
| <i>kStatus_NoTransferIn-Progress</i> | transfer has finished or no transfer in progress. |
|--------------------------------------|---------------------------------------------------|



|                        |                                |
|------------------------|--------------------------------|
| <i>kStatus_Success</i> | Successfully return the count. |
|------------------------|--------------------------------|

### 23.8.7.23 **status\_t FLEXIO\_UART\_TransferReceiveNonBlocking ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_handle\_t \* *handle*, flexio\_uart\_transfer\_t \* *xfer*, size\_t \* *receivedBytes* )**

This function receives data using the 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 ring buffer is copied and the parameter *receivedBytes* shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When 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, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to *xfer->data*. This function returns with the parameter *receivedBytes* set to 5. For the last 5 bytes, newly arrived data is saved from the *xfer->data[5]*. When 5 bytes are received, the UART driver notifies upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to *xfer->data*. When all data is received, the upper layer is notified.

#### Parameters

|                      |                                                                                         |
|----------------------|-----------------------------------------------------------------------------------------|
| <i>base</i>          | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.                              |
| <i>handle</i>        | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state. |
| <i>xfer</i>          | UART transfer structure. See <a href="#">flexio_uart_transfer_t</a> .                   |
| <i>receivedBytes</i> | Bytes received from the ring buffer directly.                                           |

#### Return values

|                                   |                                                          |
|-----------------------------------|----------------------------------------------------------|
| <i>kStatus_Success</i>            | Successfully queue the transfer into the transmit queue. |
| <i>kStatus_FLEXIO_UART-RxBusy</i> | Previous receive request is not finished.                |

### 23.8.7.24 **void FLEXIO\_UART\_TransferAbortReceive ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_handle\_t \* *handle* )**

This function aborts the receive data which was using IRQ.

## Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state. |

### 23.8.7.25 `status_t FLEXIO_UART_TransferGetReceiveCount ( FLEXIO_UART_Type * base, flexio_uart_handle_t * handle, size_t * count )`

This function gets the number of bytes received driven by interrupt.

## Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state. |
| <i>count</i>  | Number of bytes received so far by the non-blocking transaction.                        |

## Return values

|                                     |                                                   |
|-------------------------------------|---------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | transfer has finished or no transfer in progress. |
| <i>kStatus_Success</i>              | Successfully return the count.                    |

### 23.8.7.26 `void FLEXIO_UART_TransferHandleIRQ ( void * uartType, void * uartHandle )`

This function processes the FlexIO UART transmit and receives the IRQ request.

## Parameters

|                   |                                                                                         |
|-------------------|-----------------------------------------------------------------------------------------|
| <i>uartType</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.                              |
| <i>uartHandle</i> | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state. |

## 23.8.8 FlexIO eDMA UART Driver

### 23.8.8.1 Overview

#### Data Structures

- struct `flexio_uart_edma_handle_t`  
*UART eDMA handle. [More...](#)*

#### Typedefs

- typedef void(\* `flexio_uart_edma_transfer_callback_t`)(`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `status_t` status, void \*userData)  
*UART transfer callback function.*

#### Driver version

- #define `FSL_FLEXIO_UART_EDMA_DRIVER_VERSION` (`MAKE_VERSION(2, 4, 1)`)  
*FlexIO UART EDMA driver version.*

#### eDMA transactional

- `status_t FLEXIO_UART_TransferCreateHandleEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `flexio_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 FLEXIO_UART_TransferSendEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `flexio_uart_transfer_t` \*xfer)  
*Sends data using eDMA.*
- `status_t FLEXIO_UART_TransferReceiveEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `flexio_uart_transfer_t` \*xfer)  
*Receives data using eDMA.*
- void `FLEXIO_UART_TransferAbortSendEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle)  
*Aborts the sent data which using eDMA.*
- void `FLEXIO_UART_TransferAbortReceiveEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle)  
*Aborts the receive data which using eDMA.*
- `status_t FLEXIO_UART_TransferGetSendCountEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `size_t` \*count)  
*Gets the number of bytes sent out.*
- `status_t FLEXIO_UART_TransferGetReceiveCountEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `size_t` \*count)  
*Gets the number of bytes received.*

## 23.8.8.2 Data Structure Documentation

### 23.8.8.2.1 struct `_flexio_uart_edma_handle`

#### Data Fields

- `flexio_uart_edma_transfer_callback_t` `callback`  
*Callback function.*
- `void *` `userData`  
*UART callback function parameter.*
- `size_t` `txDataSizeAll`  
*Total bytes to be sent.*
- `size_t` `rxDataSizeAll`  
*Total bytes to be received.*
- `edma_handle_t *` `txEdmaHandle`  
*The eDMA TX channel used.*
- `edma_handle_t *` `rxEdmaHandle`  
*The eDMA RX channel used.*
- `uint8_t` `nbytes`  
*eDMA minor byte transfer count initially configured.*
- `volatile uint8_t` `txState`  
*TX transfer state.*
- `volatile uint8_t` `rxState`  
*RX transfer state.*

## Field Documentation

- (1) `flexio_uart_edma_transfer_callback_t flexio_uart_edma_handle_t::callback`
- (2) `void* flexio_uart_edma_handle_t::userData`
- (3) `size_t flexio_uart_edma_handle_t::txDataSizeAll`
- (4) `size_t flexio_uart_edma_handle_t::rxDataSizeAll`
- (5) `edma_handle_t* flexio_uart_edma_handle_t::txEdmaHandle`
- (6) `edma_handle_t* flexio_uart_edma_handle_t::rxEdmaHandle`
- (7) `uint8_t flexio_uart_edma_handle_t::nbytes`
- (8) `volatile uint8_t flexio_uart_edma_handle_t::txState`

### 23.8.8.3 Macro Definition Documentation

23.8.8.3.1 `#define FSL_FLEXIO_UART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 1))`

### 23.8.8.4 Typedef Documentation

23.8.8.4.1 `typedef void(* flexio_uart_edma_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_edma_handle_t *handle, status_t status, void *userData)`

### 23.8.8.5 Function Documentation

23.8.8.5.1 `status_t FLEXIO_UART_TransferCreateHandleEDMA ( FLEXIO_UART_Type * base, flexio_uart_edma_handle_t * handle, flexio_uart_edma_transfer_callback_t callback, void * userData, edma_handle_t * txEdmaHandle, edma_handle_t * rxEdmaHandle )`

## Parameters

|                     |                                                 |
|---------------------|-------------------------------------------------|
| <i>base</i>         | Pointer to <a href="#">FLEXIO_UART_Type</a> .   |
| <i>handle</i>       | Pointer to flexio_uart_edma_handle_t structure. |
| <i>callback</i>     | The callback function.                          |
| <i>userData</i>     | The parameter of the callback function.         |
| <i>rxEdmaHandle</i> | User requested DMA handle for RX DMA transfer.  |
| <i>txEdmaHandle</i> | User requested DMA handle for TX DMA transfer.  |

## Return values

|                           |                                                     |
|---------------------------|-----------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                     |
| <i>kStatus_OutOfRange</i> | The FlexIO SPI eDMA type/handle table out of range. |

### 23.8.8.5.2 status\_t FLEXIO\_UART\_TransferSendEDMA ( FLEXIO\_UART\_Type \* base, flexio\_uart\_edma\_handle\_t \* handle, flexio\_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 out, the send callback function is called.

## Parameters

|               |                                                                            |
|---------------|----------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_UART_Type</a>                                |
| <i>handle</i> | UART handle pointer.                                                       |
| <i>xfer</i>   | UART eDMA transfer structure, see <a href="#">flexio_uart_transfer_t</a> . |

## Return values

|                                   |                             |
|-----------------------------------|-----------------------------|
| <i>kStatus_Success</i>            | if succeed, others failed.  |
| <i>kStatus_FLEXIO_UART-TxBusy</i> | Previous transfer on going. |

### 23.8.8.5.3 status\_t FLEXIO\_UART\_TransferReceiveEDMA ( FLEXIO\_UART\_Type \* base, flexio\_uart\_edma\_handle\_t \* handle, flexio\_uart\_transfer\_t \* xfer )

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

## Parameters

|               |                                                                            |
|---------------|----------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_UART_Type</a>                                |
| <i>handle</i> | Pointer to flexio_uart_edma_handle_t structure                             |
| <i>xfer</i>   | UART eDMA transfer structure, see <a href="#">flexio_uart_transfer_t</a> . |

## Return values

|                            |                             |
|----------------------------|-----------------------------|
| <i>kStatus_Success</i>     | if succeed, others failed.  |
| <i>kStatus_UART_RxBusy</i> | Previous transfer on going. |

#### 23.8.8.5.4 void FLEXIO\_UART\_TransferAbortSendEDMA ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_edma\_handle\_t \* *handle* )

This function aborts sent data which using eDMA.

## Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_UART_Type</a>    |
| <i>handle</i> | Pointer to flexio_uart_edma_handle_t structure |

#### 23.8.8.5.5 void FLEXIO\_UART\_TransferAbortReceiveEDMA ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_edma\_handle\_t \* *handle* )

This function aborts the receive data which using eDMA.

## Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_UART_Type</a>    |
| <i>handle</i> | Pointer to flexio_uart_edma_handle_t structure |

#### 23.8.8.5.6 status\_t FLEXIO\_UART\_TransferGetSendCountEDMA ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_edma\_handle\_t \* *handle*, size\_t \* *count* )

This function gets the number of bytes sent out.

## Parameters

|               |                                                              |
|---------------|--------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_UART_Type</a>                  |
| <i>handle</i> | Pointer to flexio_uart_edma_handle_t structure               |
| <i>count</i>  | Number of bytes sent so far by the non-blocking transaction. |

## Return values

|                                     |                                                   |
|-------------------------------------|---------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | transfer has finished or no transfer in progress. |
| <i>kStatus_Success</i>              | Successfully return the count.                    |

### 23.8.8.5.7 status\_t FLEXIO\_UART\_TransferGetReceiveCountEDMA ( FLEXIO\_UART\_Type \* base, flexio\_uart\_edma\_handle\_t \* handle, size\_t \* count )

This function gets the number of bytes received.

## Parameters

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_UART_Type</a>                      |
| <i>handle</i> | Pointer to flexio_uart_edma_handle_t structure                   |
| <i>count</i>  | Number of bytes received so far by the non-blocking transaction. |

## Return values

|                                     |                                                   |
|-------------------------------------|---------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | transfer has finished or no transfer in progress. |
| <i>kStatus_Success</i>              | Successfully return the count.                    |



## Chapter 24

# FLEXRAM: on-chip RAM manager

### 24.1 Overview

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

The FLEXRAM module integrates the ITCM, DTCM, and OCRAM controllers, and supports parameterized RAM array and RAM array portioning.

This example code shows how to allocate RAM using the FLEXRAM driver.

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

### Data Structures

- struct `flexram_allocate_ram_t`  
*FLEXRAM allocate ocram, itcm, dtcm size. [More...](#)*

### Macros

- `#define FLEXRAM_ECC_ERROR_DETAILED_INFO 0U` /\* Define to zero means get raw ECC error information, which needs parse it by user. \*/  
*Get ECC error detailed information.*

### Enumerations

- enum {  
    `kFLEXRAM_BankNotUsed` = 0U,  
    `kFLEXRAM_BankOCRAM` = 1U,  
    `kFLEXRAM_BankDTCM` = 2U,  
    `kFLEXRAM_BankITCM` = 3U }  
*FLEXRAM bank type.*
- enum `flexram_bank_allocate_src_t` {  
    `kFLEXRAM_BankAllocateThroughHardwareFuse` = 0U,  
    `kFLEXRAM_BankAllocateThroughBankCfg` = 1U }  
*FLEXRAM bank allocate source.*
- enum {  
    `kFLEXRAM_Read` = 0U,  
    `kFLEXRAM_Write` = 1U }  
*Flexram write/read selection.*
- enum {  
    `kFLEXRAM_OCRAMAccessError` = FLEXRAM\_INT\_STATUS\_OCRAM\_ERR\_STATUS\_MASK,  
    `kFLEXRAM_DTCMAccessError` = FLEXRAM\_INT\_STATUS\_DTCM\_ERR\_STATUS\_MASK,  
    `kFLEXRAM_ITCMAccessError` = FLEXRAM\_INT\_STATUS\_ITCM\_ERR\_STATUS\_MASK,

```

kFLEXRAM_InterruptStatusAll }
 Interrupt status flag mask.
• enum flexram_tcm_access_mode_t {
 kFLEXRAM_TCMAccessFastMode = 0U,
 kFLEXRAM_TCMAccessWaitMode = 1U }
 FLEXRAM TCM access mode.
• enum {
 kFLEXRAM_TCMSize32KB = 32 * 1024U,
 kFLEXRAM_TCMSize64KB = 64 * 1024U,
 kFLEXRAM_TCMSize128KB = 128 * 1024U,
 kFLEXRAM_TCMSize256KB = 256 * 1024U,
 kFLEXRAM_TCMSize512KB = 512 * 1024U }
 FLEXRAM TCM support size.

```

## Functions

- `status_t FLEXRAM_AllocateRam (flexram_allocate_ram_t *config)`  
*FLEXRAM allocate on-chip ram for OCRAM,ITCM,DTCM This function is independent of FLEXRAM\_Init, it can be called directly if ram re-allocate is needed.*
- `static void FLEXRAM_SetAllocateRamSrc (flexram_bank_allocate_src_t src)`  
*FLEXRAM set allocate on-chip ram source.*
- `static void FLEXRAM_SetTCMReadAccessMode (FLEXRAM_Type *base, flexram_tcm_access_mode_t mode)`  
*FLEXRAM module sets TCM read access mode.*
- `static void FLEXRAM_SetTCMWriteAccessMode (FLEXRAM_Type *base, flexram_tcm_access_mode_t mode)`  
*FLEXRAM module set TCM write access mode.*
- `static void FLEXRAM_EnableForceRamClockOn (FLEXRAM_Type *base, bool enable)`  
*FLEXRAM module force ram clock on.*

## Driver version

- `#define FSL_SOC_FLEXRAM_ALLOCATE_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`  
*SOC\_FLEXRAM\_ALLOCATE driver version 2.0.2.*

## Driver version

- `#define FSL_FLEXRAM_DRIVER_VERSION (MAKE_VERSION(2U, 3U, 0U))`  
*Driver version.*

## Initialization and de-initialization

- `void FLEXRAM_Init (FLEXRAM_Type *base)`  
*FLEXRAM module initialization function.*
- `void FLEXRAM_Deinit (FLEXRAM_Type *base)`  
*De-initializes the FLEXRAM.*

## Status

- `static uint32_t FLEXRAM_GetInterruptStatus (FLEXRAM_Type *base)`

- *FLEXRAM module gets interrupt status.*  
static void [FLEXRAM\\_ClearInterruptStatus](#) (FLEXRAM\_Type \*base, uint32\_t status)
- *FLEXRAM module clears interrupt status.*  
static void [FLEXRAM\\_EnableInterruptStatus](#) (FLEXRAM\_Type \*base, uint32\_t status)
- *FLEXRAM module enables interrupt status.*  
static void [FLEXRAM\\_DisableInterruptStatus](#) (FLEXRAM\_Type \*base, uint32\_t status)
- *FLEXRAM module disable interrupt status.*

### Interrupts

- static void [FLEXRAM\\_EnableInterruptSignal](#) (FLEXRAM\_Type \*base, uint32\_t status)  
*FLEXRAM module enables interrupt.*
- static void [FLEXRAM\\_DisableInterruptSignal](#) (FLEXRAM\_Type \*base, uint32\_t status)  
*FLEXRAM module disables interrupt.*

## 24.2 Data Structure Documentation

### 24.2.1 struct flexram\_allocate\_ram\_t

#### Data Fields

- const uint8\_t [ocramBankNum](#)  
*ocram banknumber which the SOC support*
- const uint8\_t [dtdcmBankNum](#)  
*dtdcm bank number to allocate, the number should be power of 2*
- const uint8\_t [itcmBankNum](#)  
*itcm bank number to allocate, the number should be power of 2*

## 24.3 Macro Definition Documentation

24.3.1 **#define FSL\_SOC\_FLEXRAM\_ALLOCATE\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))**

24.3.2 **#define FSL\_FLEXRAM\_DRIVER\_VERSION (MAKE\_VERSION(2U, 3U, 0U))**

24.3.3 **#define FLEXRAM\_ECC\_ERROR\_DETAILED\_INFO 0U /\* Define to zero means get raw ECC error information, which needs parse it by user. \*/**

## 24.4 Enumeration Type Documentation

### 24.4.1 anonymous enum

Enumerator

*kFLEXRAM\_BankNotUsed* bank is not used  
*kFLEXRAM\_BankOCRAM* bank is OCRAM  
*kFLEXRAM\_BankDTCM* bank is DTCM

*kFLEXRAM\_BankITCM* bank is ITCM

#### 24.4.2 enum flexram\_bank\_allocate\_src\_t

Enumerator

*kFLEXRAM\_BankAllocateThroughHardwareFuse* allocate ram through hardware fuse value  
*kFLEXRAM\_BankAllocateThroughBankCfg* allocate ram through FLEXRAM\_BANK\_CFG

#### 24.4.3 anonymous enum

Enumerator

*kFLEXRAM\_Read* read  
*kFLEXRAM\_Write* write

#### 24.4.4 anonymous enum

Enumerator

*kFLEXRAM\_OCRAccessError* OCRAM accesses unallocated address.  
*kFLEXRAM\_DTCMAccessError* DTCM accesses unallocated address.  
*kFLEXRAM\_ITCMAccessError* ITCM accesses unallocated address.  
*kFLEXRAM\_InterruptStatusAll* all the interrupt status mask

#### 24.4.5 enum flexram\_tcm\_access\_mode\_t

Fast access mode expected to be finished in 1-cycle; Wait access mode expected to be finished in 2-cycle. Wait access mode is a feature of the flexram and it should be used when the CPU clock is too fast to finish TCM access in 1-cycle. Normally, fast mode is the default mode, the efficiency of the TCM access will be better.

Enumerator

*kFLEXRAM\_TCMAccessFastMode* fast access mode  
*kFLEXRAM\_TCMAccessWaitMode* wait access mode

#### 24.4.6 anonymous enum

Enumerator

*kFLEXRAM\_TCMSize32KB* TCM total size be 32KB.

***kFLEXRAM\_TCMSize64KB*** TCM total size be 64KB.  
***kFLEXRAM\_TCMSize128KB*** TCM total size be 128KB.  
***kFLEXRAM\_TCMSize256KB*** TCM total size be 256KB.  
***kFLEXRAM\_TCMSize512KB*** TCM total size be 512KB.

## 24.5 Function Documentation

### 24.5.1 `status_t FLEXRAM_AllocateRam ( flexram_allocate_ram_t * config )`

Parameters

|               |                         |
|---------------|-------------------------|
| <i>config</i> | allocate configuration. |
|---------------|-------------------------|

Return values

|                                |                         |                        |                  |
|--------------------------------|-------------------------|------------------------|------------------|
| <i>kStatus_InvalidArgument</i> | the argument is invalid | <i>kStatus_Success</i> | allocate success |
|--------------------------------|-------------------------|------------------------|------------------|

### 24.5.2 `static void FLEXRAM_SetAllocateRamSrc ( flexram_bank_allocate_src_t src ) [inline], [static]`

Parameters

|            |                                  |
|------------|----------------------------------|
| <i>src</i> | bank config source select value. |
|------------|----------------------------------|

### 24.5.3 `void FLEXRAM_Init ( FLEXRAM_Type * base )`

Parameters

|             |                       |
|-------------|-----------------------|
| <i>base</i> | FLEXRAM base address. |
|-------------|-----------------------|

### 24.5.4 `static uint32_t FLEXRAM_GetInterruptStatus ( FLEXRAM_Type * base ) [inline], [static]`

Parameters

---

|             |                       |
|-------------|-----------------------|
| <i>base</i> | FLEXRAM base address. |
|-------------|-----------------------|

**24.5.5 static void FLEXRAM\_ClearInterruptStatus ( FLEXRAM\_Type \* *base*,  
uint32\_t *status* ) [inline], [static]**

Parameters

|               |                       |
|---------------|-----------------------|
| <i>base</i>   | FLEXRAM base address. |
| <i>status</i> | Status to be cleared. |

**24.5.6 static void FLEXRAM\_EnableInterruptStatus ( FLEXRAM\_Type \* *base*,  
uint32\_t *status* ) [inline], [static]**

Parameters

|               |                       |
|---------------|-----------------------|
| <i>base</i>   | FLEXRAM base address. |
| <i>status</i> | Status to be enabled. |

**24.5.7 static void FLEXRAM\_DisableInterruptStatus ( FLEXRAM\_Type \* *base*,  
uint32\_t *status* ) [inline], [static]**

Parameters

|               |                        |
|---------------|------------------------|
| <i>base</i>   | FLEXRAM base address.  |
| <i>status</i> | Status to be disabled. |

**24.5.8 static void FLEXRAM\_EnableInterruptSignal ( FLEXRAM\_Type \* *base*,  
uint32\_t *status* ) [inline], [static]**

Parameters

---

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | FLEXRAM base address.           |
| <i>status</i> | Status interrupt to be enabled. |

**24.5.9 static void FLEXRAM\_DisableInterruptSignal ( FLEXRAM\_Type \* *base*, uint32\_t *status* ) [inline], [static]**

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXRAM base address.            |
| <i>status</i> | Status interrupt to be disabled. |

**24.5.10 static void FLEXRAM\_SetTCMReadAccessMode ( FLEXRAM\_Type \* *base*, flexram\_tcm\_access\_mode\_t *mode* ) [inline], [static]**

Parameters

|             |                       |
|-------------|-----------------------|
| <i>base</i> | FLEXRAM base address. |
| <i>mode</i> | Access mode.          |

**24.5.11 static void FLEXRAM\_SetTCMWriteAccessMode ( FLEXRAM\_Type \* *base*, flexram\_tcm\_access\_mode\_t *mode* ) [inline], [static]**

Parameters

|             |                       |
|-------------|-----------------------|
| <i>base</i> | FLEXRAM base address. |
| <i>mode</i> | Access mode.          |

**24.5.12 static void FLEXRAM\_EnableForceRamClockOn ( FLEXRAM\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FLEXRAM base address.             |
| <i>enable</i> | Enable or disable clock force on. |



## Chapter 25

# FLEXSPI: Flexible Serial Peripheral Interface Driver

### 25.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Flexible Serial Peripheral Interface (FLEXSPI) module of MCUXpresso SDK/i.MX devices.

FLEXSPI driver includes functional APIs and interrupt/EDMA non-blocking transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for FLEXSPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FLEXSPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. FLEXSPI functional operation groups provide the functional API set.

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the `flexspi_handle_t`/`flexspi_edma_handle_t` as the second parameter. Initialize the handle for interrupt non-blocking transfer by calling the `FLEXSPI_TransferCreateHandle` API. Initialize the handle for interrupt non-blocking transfer by calling the `FLEXSPI_TransferCreateHandleEDMA` API.

Transactional APIs support asynchronous transfer. This means that the functions `FLEXSPI_TransferNonBlocking()` and `FLEXSPI_TransferEDMA()` set up data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_FLEXSPI_Idle` status.

### Modules

- [FLEXSPI eDMA Driver](#)

### Data Structures

- struct `flexspi_config_t`  
*FLEXSPI configuration structure. [More...](#)*
- struct `flexspi_device_config_t`  
*External device configuration items. [More...](#)*
- struct `flexspi_transfer_t`  
*Transfer structure for FLEXSPI. [More...](#)*
- struct `flexspi_handle_t`  
*Transfer handle structure for FLEXSPI. [More...](#)*

### Macros

- #define `FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)`  
*Formula to form FLEXSPI instructions in LUT table.*

## Typedefs

- typedef void(\* flexspi\_transfer\_callback\_t )(FLEXSPI\_Type \*base, flexspi\_handle\_t \*handle, status\_t status, void \*userData)  
*FLEXSPI transfer callback function.*

## Enumerations

- enum {  
kStatus\_FLEXSPI\_Busy = MAKE\_STATUS(kStatusGroup\_FLEXSPI, 0),  
kStatus\_FLEXSPI\_SequenceExecutionTimeout = MAKE\_STATUS(kStatusGroup\_FLEXSPI, 1),  
kStatus\_FLEXSPI\_IpCommandSequenceError = MAKE\_STATUS(kStatusGroup\_FLEXSPI, 2),  
kStatus\_FLEXSPI\_IpCommandGrantTimeout = MAKE\_STATUS(kStatusGroup\_FLEXSPI, 3) }  
*Status structure of FLEXSPI.*

- enum {  
kFLEXSPI\_Command\_STOP = 0x00U,  
kFLEXSPI\_Command\_SDR = 0x01U,  
kFLEXSPI\_Command\_RADDR\_SDR = 0x02U,  
kFLEXSPI\_Command\_CADDR\_SDR = 0x03U,  
kFLEXSPI\_Command\_MODE1\_SDR = 0x04U,  
kFLEXSPI\_Command\_MODE2\_SDR = 0x05U,  
kFLEXSPI\_Command\_MODE4\_SDR = 0x06U,  
kFLEXSPI\_Command\_MODE8\_SDR = 0x07U,  
kFLEXSPI\_Command\_WRITE\_SDR = 0x08U,  
kFLEXSPI\_Command\_READ\_SDR = 0x09U,  
kFLEXSPI\_Command\_LEARN\_SDR = 0x0AU,  
kFLEXSPI\_Command\_DATSZ\_SDR = 0x0BU,  
kFLEXSPI\_Command\_DUMMY\_SDR = 0x0CU,  
kFLEXSPI\_Command\_DUMMY\_RWDS\_SDR = 0x0DU,  
kFLEXSPI\_Command\_DDR = 0x21U,  
kFLEXSPI\_Command\_RADDR\_DDR = 0x22U,  
kFLEXSPI\_Command\_CADDR\_DDR = 0x23U,  
kFLEXSPI\_Command\_MODE1\_DDR = 0x24U,  
kFLEXSPI\_Command\_MODE2\_DDR = 0x25U,  
kFLEXSPI\_Command\_MODE4\_DDR = 0x26U,  
kFLEXSPI\_Command\_MODE8\_DDR = 0x27U,  
kFLEXSPI\_Command\_WRITE\_DDR = 0x28U,  
kFLEXSPI\_Command\_READ\_DDR = 0x29U,  
kFLEXSPI\_Command\_LEARN\_DDR = 0x2AU,  
kFLEXSPI\_Command\_DATSZ\_DDR = 0x2BU,  
kFLEXSPI\_Command\_DUMMY\_DDR = 0x2CU,  
kFLEXSPI\_Command\_DUMMY\_RWDS\_DDR = 0x2DU,  
kFLEXSPI\_Command\_JUMP\_ON\_CS = 0x1FU }  
*CMD definition of FLEXSPI, use to form LUT instruction, \_flexspi\_command.*
- enum flexspi\_pad\_t {

```

kFLEXSPI_1PAD = 0x00U,
kFLEXSPI_2PAD = 0x01U,
kFLEXSPI_4PAD = 0x02U,
kFLEXSPI_8PAD = 0x03U }

```

*pad definition of FLEXSPI, use to form LUT instruction.*

- enum flexspi\_flags\_t {
 

```

kFLEXSPI_SequenceExecutionTimeoutFlag = FLEXSPI_INTEN_SEQTIMEOUTEN_MASK,
kFLEXSPI_AhbBusTimeoutFlag = FLEXSPI_INTEN_AHBBUSTIMEOUTEN_MASK,
kFLEXSPI_SckStoppedBecauseTxEmptyFlag,
kFLEXSPI_SckStoppedBecauseRxFullFlag,
kFLEXSPI_DataLearningFailedFlag = FLEXSPI_INTEN_DATALEARNFAILEN_MASK,
kFLEXSPI_IpTxFifoWatermarkEmptyFlag = FLEXSPI_INTEN_IPTXWEEN_MASK,
kFLEXSPI_IpRxFifoWatermarkAvailableFlag = FLEXSPI_INTEN_IPRXWAEN_MASK,
kFLEXSPI_AhbCommandSequenceErrorFlag,
kFLEXSPI_IpCommandSequenceErrorFlag = FLEXSPI_INTEN_IPCMDERREN_MASK,
kFLEXSPI_AhbCommandGrantTimeoutFlag,
kFLEXSPI_IpCommandGrantTimeoutFlag,
kFLEXSPI_IpCommandExecutionDoneFlag,
kFLEXSPI_AllInterruptFlags = 0xFFFU }

```

*FLEXSPI interrupt status flags.*

- enum flexspi\_read\_sample\_clock\_t {
 

```

kFLEXSPI_ReadSampleClkLoopbackInternally = 0x0U,
kFLEXSPI_ReadSampleClkLoopbackFromDqsPad = 0x1U,
kFLEXSPI_ReadSampleClkLoopbackFromSckPad = 0x2U,
kFLEXSPI_ReadSampleClkExternalInputFromDqsPad = 0x3U }

```

*FLEXSPI sample clock source selection for Flash Reading.*

- enum flexspi\_cs\_interval\_cycle\_unit\_t {
 

```

kFLEXSPI_CsIntervalUnit1SckCycle = 0x0U,
kFLEXSPI_CsIntervalUnit256SckCycle = 0x1U }

```

*FLEXSPI interval unit for flash device select.*

- enum flexspi\_ahb\_write\_wait\_unit\_t {
 

```

kFLEXSPI_AhbWriteWaitUnit2AhbCycle = 0x0U,
kFLEXSPI_AhbWriteWaitUnit8AhbCycle = 0x1U,
kFLEXSPI_AhbWriteWaitUnit32AhbCycle = 0x2U,
kFLEXSPI_AhbWriteWaitUnit128AhbCycle = 0x3U,
kFLEXSPI_AhbWriteWaitUnit512AhbCycle = 0x4U,
kFLEXSPI_AhbWriteWaitUnit2048AhbCycle = 0x5U,
kFLEXSPI_AhbWriteWaitUnit8192AhbCycle = 0x6U,
kFLEXSPI_AhbWriteWaitUnit32768AhbCycle = 0x7U }

```

*FLEXSPI AHB wait interval unit for writing.*

- enum flexspi\_ip\_error\_code\_t {

```

kFLEXSPI_IpCmdErrorNoError = 0x0U,
kFLEXSPI_IpCmdErrorJumpOnCsInIpCmd = 0x2U,
kFLEXSPI_IpCmdErrorUnknownOpCode = 0x3U,
kFLEXSPI_IpCmdErrorSdrDummyInDdrSequence = 0x4U,
kFLEXSPI_IpCmdErrorDdrDummyInSdrSequence = 0x5U,
kFLEXSPI_IpCmdErrorInvalidAddress = 0x6U,
kFLEXSPI_IpCmdErrorSequenceExecutionTimeout = 0xEU,
kFLEXSPI_IpCmdErrorFlashBoundaryAcrosss = 0xFU }

```

*Error Code when IP command Error detected.*

- enum flexspi\_ahb\_error\_code\_t {

```

kFLEXSPI_AhbCmdErrorNoError = 0x0U,
kFLEXSPI_AhbCmdErrorJumpOnCsInWriteCmd = 0x2U,
kFLEXSPI_AhbCmdErrorUnknownOpCode = 0x3U,
kFLEXSPI_AhbCmdErrorSdrDummyInDdrSequence = 0x4U,
kFLEXSPI_AhbCmdErrorDdrDummyInSdrSequence = 0x5U,
kFLEXSPI_AhbCmdSequenceExecutionTimeout = 0x6U }

```

*Error Code when AHB command Error detected.*

- enum flexspi\_port\_t {

```

kFLEXSPI_PortA1 = 0x0U,
kFLEXSPI_PortA2,
kFLEXSPI_PortB1,
kFLEXSPI_PortB2 }

```

*FLEXSPI operation port select.*

- enum flexspi\_arb\_command\_source\_t

*Trigger source of current command sequence granted by arbitrator.*
- enum flexspi\_command\_type\_t {

```

kFLEXSPI_Command,
kFLEXSPI_Config }

```

*Command type.*

## Driver version

- #define FSL\_FLEXSPI\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 1))

*FLEXSPI driver version.*

## Initialization and deinitialization

- uint32\_t FLEXSPI\_GetInstance (FLEXSPI\_Type \*base)

*Get the instance number for FLEXSPI.*
- status\_t FLEXSPI\_CheckAndClearError (FLEXSPI\_Type \*base, uint32\_t status)

*Check and clear IP command execution errors.*
- void FLEXSPI\_Init (FLEXSPI\_Type \*base, const flexspi\_config\_t \*config)

*Initializes the FLEXSPI module and internal state.*
- void FLEXSPI\_GetDefaultConfig (flexspi\_config\_t \*config)

*Gets default settings for FLEXSPI.*
- void FLEXSPI\_Deinit (FLEXSPI\_Type \*base)

*Deinitializes the FLEXSPI module.*

- void [FLEXSPI\\_UpdateDllValue](#) (FLEXSPI\_Type \*base, flexspi\_device\_config\_t \*config, flexspi\_port\_t port)  
*Update FLEXSPI DLL value depending on currently flexspi root clock.*
- void [FLEXSPI\\_SetFlashConfig](#) (FLEXSPI\_Type \*base, flexspi\_device\_config\_t \*config, flexspi\_port\_t port)  
*Configures the connected device parameter.*
- static void [FLEXSPI\\_SoftwareReset](#) (FLEXSPI\_Type \*base)  
*Software reset for the FLEXSPI logic.*
- static void [FLEXSPI\\_Enable](#) (FLEXSPI\_Type \*base, bool enable)  
*Enables or disables the FLEXSPI module.*

## Interrupts

- static void [FLEXSPI\\_EnableInterrupts](#) (FLEXSPI\_Type \*base, uint32\_t mask)  
*Enables the FLEXSPI interrupts.*
- static void [FLEXSPI\\_DisableInterrupts](#) (FLEXSPI\_Type \*base, uint32\_t mask)  
*Disable the FLEXSPI interrupts.*

## DMA control

- static void [FLEXSPI\\_EnableTxDMA](#) (FLEXSPI\_Type \*base, bool enable)  
*Enables or disables FLEXSPI IP Tx FIFO DMA requests.*
- static void [FLEXSPI\\_EnableRxDMA](#) (FLEXSPI\_Type \*base, bool enable)  
*Enables or disables FLEXSPI IP Rx FIFO DMA requests.*
- static uint32\_t [FLEXSPI\\_GetTxFifoAddress](#) (FLEXSPI\_Type \*base)  
*Gets FLEXSPI IP tx fifo address for DMA transfer.*
- static uint32\_t [FLEXSPI\\_GetRxFifoAddress](#) (FLEXSPI\_Type \*base)  
*Gets FLEXSPI IP rx fifo address for DMA transfer.*

## FIFO control

- static void [FLEXSPI\\_ResetFifos](#) (FLEXSPI\_Type \*base, bool txFifo, bool rxFifo)  
*Clears the FLEXSPI IP FIFO logic.*
- static void [FLEXSPI\\_GetFifoCounts](#) (FLEXSPI\_Type \*base, size\_t \*txCount, size\_t \*rxCount)  
*Gets the valid data entries in the FLEXSPI FIFOs.*

## Status

- static uint32\_t [FLEXSPI\\_GetInterruptStatusFlags](#) (FLEXSPI\_Type \*base)  
*Get the FLEXSPI interrupt status flags.*
- static void [FLEXSPI\\_ClearInterruptStatusFlags](#) (FLEXSPI\_Type \*base, uint32\_t mask)  
*Get the FLEXSPI interrupt status flags.*
- static void [FLEXSPI\\_GetDataLearningPhase](#) (FLEXSPI\_Type \*base, uint8\_t \*portAPhase, uint8\_t \*portBPhase)  
*Gets the sampling clock phase selection after Data Learning.*
- static flexspi\_arb\_command\_source\_t [FLEXSPI\\_GetArbitratorCommandSource](#) (FLEXSPI\_Type \*base)  
*Gets the trigger source of current command sequence granted by arbitrator.*
- static flexspi\_ip\_error\_code\_t [FLEXSPI\\_GetIPCommandErrorCode](#) (FLEXSPI\_Type \*base, uint8\_t \*index)

- *Gets the error code when IP command error detected.*
- static `flexspi_ahb_error_code_t FLEXSPI_GetAHBCommandErrorCode` (FLEXSPI\_Type \*base, uint8\_t \*index)
  - *Gets the error code when AHB command error detected.*
- static bool `FLEXSPI_GetBusIdleStatus` (FLEXSPI\_Type \*base)
  - *Returns whether the bus is idle.*

## Bus Operations

- void `FLEXSPI_UpdateRxSampleClock` (FLEXSPI\_Type \*base, flexspi\_read\_sample\_clock\_t clockSource)
  - *Update read sample clock source.*
- static void `FLEXSPI_EnableIPParallelMode` (FLEXSPI\_Type \*base, bool enable)
  - *Enables/disables the FLEXSPI IP command parallel mode.*
- static void `FLEXSPI_EnableAHBParallelMode` (FLEXSPI\_Type \*base, bool enable)
  - *Enables/disables the FLEXSPI AHB command parallel mode.*
- void `FLEXSPI_UpdateLUT` (FLEXSPI\_Type \*base, uint32\_t index, const uint32\_t \*cmd, uint32\_t count)
  - *Updates the LUT table.*
- static void `FLEXSPI_WriteData` (FLEXSPI\_Type \*base, uint32\_t data, uint8\_t fifoIndex)
  - *Writes data into FIFO.*
- static uint32\_t `FLEXSPI_ReadData` (FLEXSPI\_Type \*base, uint8\_t fifoIndex)
  - *Receives data from data FIFO.*
- `status_t FLEXSPI_WriteBlocking` (FLEXSPI\_Type \*base, uint8\_t \*buffer, size\_t size)
  - *Sends a buffer of data bytes using blocking method.*
- `status_t FLEXSPI_ReadBlocking` (FLEXSPI\_Type \*base, uint8\_t \*buffer, size\_t size)
  - *Receives a buffer of data bytes using a blocking method.*
- `status_t FLEXSPI_TransferBlocking` (FLEXSPI\_Type \*base, flexspi\_transfer\_t \*xfer)
  - *Execute command to transfer a buffer data bytes using a blocking method.*

## Transactional

- void `FLEXSPI_TransferCreateHandle` (FLEXSPI\_Type \*base, flexspi\_handle\_t \*handle, flexspi\_transfer\_callback\_t callback, void \*userData)
  - *Initializes the FLEXSPI handle which is used in transactional functions.*
- `status_t FLEXSPI_TransferNonBlocking` (FLEXSPI\_Type \*base, flexspi\_handle\_t \*handle, flexspi\_transfer\_t \*xfer)
  - *Performs a interrupt non-blocking transfer on the FLEXSPI bus.*
- `status_t FLEXSPI_TransferGetCount` (FLEXSPI\_Type \*base, flexspi\_handle\_t \*handle, size\_t \*count)
  - *Gets the master transfer status during a interrupt non-blocking transfer.*
- void `FLEXSPI_TransferAbort` (FLEXSPI\_Type \*base, flexspi\_handle\_t \*handle)
  - *Aborts an interrupt non-blocking transfer early.*
- void `FLEXSPI_TransferHandleIRQ` (FLEXSPI\_Type \*base, flexspi\_handle\_t \*handle)
  - *Master interrupt handler.*



## 25.2 Data Structure Documentation

### 25.2.1 struct flexspi\_config\_t

#### Data Fields

- [flexspi\\_read\\_sample\\_clock\\_t rxSampleClock](#)  
*Sample Clock source selection for Flash Reading.*
- bool [enableSckFreeRunning](#)  
*Enable/disable SCK output free-running.*
- bool [enableCombination](#)  
*Enable/disable combining PORT A and B Data Pins (SIOA[3:0] and SIOB[3:0]) to support Flash Octal mode.*
- bool [enableDoze](#)  
*Enable/disable doze mode support.*
- bool [enableHalfSpeedAccess](#)  
*Enable/disable divide by 2 of the clock for half speed commands.*
- bool [enableSckBDiffOpt](#)  
*Enable/disable SCKB pad use as SCKA differential clock output, when enable, Port B flash access is not available.*
- bool [enableSameConfigForAll](#)  
*Enable/disable same configuration for all connected devices when enabled, same configuration in FLASHA1CRx is applied to all.*
- uint16\_t [seqTimeoutCycle](#)  
*Timeout wait cycle for command sequence execution, timeout after  $ahbGrantTimeoutCycle*1024$  serial root clock cycles.*
- uint8\_t [ipGrantTimeoutCycle](#)  
*Timeout wait cycle for IP command grant, timeout after  $ipGrantTimeoutCycle*1024$  AHB clock cycles.*
- uint8\_t [txWatermark](#)  
*FLEXSPI IP transmit watermark value.*
- uint8\_t [rxWatermark](#)  
*FLEXSPI receive watermark value.*
- bool [enableAHBWriteIpTxFifo](#)  
*Enable AHB bus write access to IP TX FIFO.*
- bool [enableAHBWriteIpRxFifo](#)  
*Enable AHB bus write access to IP RX FIFO.*
- uint8\_t [ahbGrantTimeoutCycle](#)  
*Timeout wait cycle for AHB command grant, timeout after  $ahbGrantTimeoutCycle*1024$  AHB clock cycles.*
- uint16\_t [ahbBusTimeoutCycle](#)  
*Timeout wait cycle for AHB read/write access, timeout after  $ahbBusTimeoutCycle*1024$  AHB clock cycles.*
- uint8\_t [resumeWaitCycle](#)  
*Wait cycle for idle state before suspended command sequence resume, timeout after  $ahbBusTimeoutCycle$  AHB clock cycles.*
- flexspi\_ahbBuffer\_config\_t [buffer](#) [FSL\_FEATURE\_FLEXSPI\_AHB\_BUFFER\_COUNT]  
*AHB buffer size.*
- bool [enableClearAHBBufferOpt](#)  
*Enable/disable automatically clean AHB RX Buffer and TX Buffer*

- when FLEXSPI returns STOP mode ACK.*
- bool `enableReadAddressOpt`  
*Enable/disable remove AHB read burst start address alignment limitation.*
- bool `enableAHBPrefetch`  
*Enable/disable AHB read prefetch feature, when enabled, FLEXSPI will fetch more data than current AHB burst.*
- bool `enableAHBBufferable`  
*Enable/disable AHB bufferable write access support, when enabled, FLEXSPI return before waiting for command execution finished.*
- bool `enableAHBCachable`  
*Enable AHB bus cachable read access support.*



## Field Documentation

- (1) `flexspi_read_sample_clock_t flexspi_config_t::rxSampleClock`
- (2) `bool flexspi_config_t::enableSckFreeRunning`
- (3) `bool flexspi_config_t::enableCombination`
- (4) `bool flexspi_config_t::enableDoze`
- (5) `bool flexspi_config_t::enableHalfSpeedAccess`
- (6) `bool flexspi_config_t::enableSckBDiffOpt`
- (7) `bool flexspi_config_t::enableSameConfigForAll`
- (8) `uint16_t flexspi_config_t::seqTimeoutCycle`
- (9) `uint8_t flexspi_config_t::ipGrantTimeoutCycle`
- (10) `uint8_t flexspi_config_t::txWatermark`
- (11) `uint8_t flexspi_config_t::rxWatermark`
- (12) `bool flexspi_config_t::enableAHBWriteIpTxFifo`
- (13) `bool flexspi_config_t::enableAHBWriteIpRxFifo`
- (14) `uint8_t flexspi_config_t::ahbGrantTimeoutCycle`
- (15) `uint16_t flexspi_config_t::ahbBusTimeoutCycle`
- (16) `uint8_t flexspi_config_t::resumeWaitCycle`
- (17) `flexspi_ahbBuffer_config_t flexspi_config_t::buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT]`
- (18) `bool flexspi_config_t::enableClearAHBBufferOpt`
- (19) `bool flexspi_config_t::enableReadAddressOpt`

when enable, there is no AHB read burst start address alignment limitation.

- (20) `bool flexspi_config_t::enableAHBPrefetch`
- (21) `bool flexspi_config_t::enableAHBBufferable`
- (22) `bool flexspi_config_t::enableAHBCachable`

## 25.2.2 struct flexspi\_device\_config\_t

### Data Fields

- `uint32_t flexspiRootClk`  
*FLEXSPI serial root clock.*
- `bool isSck2Enabled`  
*FLEXSPI use SCK2.*
- `uint32_t flashSize`  
*Flash size in KByte.*
- `flexspi_cs_interval_cycle_unit_t CSIntervalUnit`  
*CS interval unit, 1 or 256 cycle.*
- `uint16_t CSInterval`  
*CS line assert interval, multiply CS interval unit to get the CS line assert interval cycles.*
- `uint8_t CSHoldTime`  
*CS line hold time.*
- `uint8_t CSSetupTime`  
*CS line setup time.*
- `uint8_t dataValidTime`  
*Data valid time for external device.*
- `uint8_t columnSpace`  
*Column space size.*
- `bool enableWordAddress`  
*If enable word address.*
- `uint8_t AWRSeqIndex`  
*Sequence ID for AHB write command.*
- `uint8_t AWRSeqNumber`  
*Sequence number for AHB write command.*
- `uint8_t ARDSeqIndex`  
*Sequence ID for AHB read command.*
- `uint8_t ARDSeqNumber`  
*Sequence number for AHB read command.*
- `flexspi_ahb_write_wait_unit_t AHBWriteWaitUnit`  
*AHB write wait unit.*
- `uint16_t AHBWriteWaitInterval`  
*AHB write wait interval, multiply AHB write interval unit to get the AHB write wait cycles.*
- `bool enableWriteMask`  
*Enable/Disable FLEXSPI drive DQS pin as write mask when writing to external device.*

## Field Documentation

- (1) `uint32_t flexspi_device_config_t::flexspiRootClk`
- (2) `bool flexspi_device_config_t::isSck2Enabled`
- (3) `uint32_t flexspi_device_config_t::flashSize`
- (4) `flexspi_cs_interval_cycle_unit_t flexspi_device_config_t::CSIntervalUnit`
- (5) `uint16_t flexspi_device_config_t::CSInterval`
- (6) `uint8_t flexspi_device_config_t::CSHoldTime`
- (7) `uint8_t flexspi_device_config_t::CSSetupTime`
- (8) `uint8_t flexspi_device_config_t::dataValidTime`
- (9) `uint8_t flexspi_device_config_t::columnspace`
- (10) `bool flexspi_device_config_t::enableWordAddress`
- (11) `uint8_t flexspi_device_config_t::AWRSeqIndex`
- (12) `uint8_t flexspi_device_config_t::AWRSeqNumber`
- (13) `uint8_t flexspi_device_config_t::ARDSeqIndex`
- (14) `uint8_t flexspi_device_config_t::ARDSeqNumber`
- (15) `flexspi_ahb_write_wait_unit_t flexspi_device_config_t::AHBWriteWaitUnit`
- (16) `uint16_t flexspi_device_config_t::AHBWriteWaitInterval`
- (17) `bool flexspi_device_config_t::enableWriteMask`

## 25.2.3 struct flexspi\_transfer\_t

## Data Fields

- `uint32_t deviceAddress`  
*Operation device address.*
- `flexspi_port_t port`  
*Operation port.*
- `flexspi_command_type_t cmdType`  
*Execution command type.*
- `uint8_t seqIndex`  
*Sequence ID for command.*
- `uint8_t SeqNumber`  
*Sequence number for command.*

- `uint32_t * data`  
*Data buffer.*
- `size_t dataSize`  
*Data size in bytes.*

#### Field Documentation

- (1) `uint32_t flexspi_transfer_t::deviceAddress`
- (2) `flexspi_port_t flexspi_transfer_t::port`
- (3) `flexspi_command_type_t flexspi_transfer_t::cmdType`
- (4) `uint8_t flexspi_transfer_t::seqIndex`
- (5) `uint8_t flexspi_transfer_t::SeqNumber`
- (6) `uint32_t* flexspi_transfer_t::data`
- (7) `size_t flexspi_transfer_t::dataSize`

### 25.2.4 struct flexspi\_handle

#### Data Fields

- `uint32_t state`  
*Internal state for FLEXSPI transfer.*
- `uint8_t * data`  
*Data buffer.*
- `size_t dataSize`  
*Remaining Data size in bytes.*
- `size_t transferTotalSize`  
*Total Data size in bytes.*
- `flexspi_transfer_callback_t completionCallback`  
*Callback for users while transfer finish or error occurred.*
- `void * userData`  
*FLEXSPI callback function parameter.*

#### Field Documentation

- (1) `uint8_t* flexspi_handle_t::data`
- (2) `size_t flexspi_handle_t::dataSize`
- (3) `size_t flexspi_handle_t::transferTotalSize`
- (4) `void* flexspi_handle_t::userData`

## 25.3 Macro Definition Documentation

**25.3.1 #define FSL\_FLEXSPI\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 1))**

**25.3.2 #define FLEXSPI\_LUT\_SEQ( cmd0, pad0, op0, cmd1, pad1, op1 )**

**Value:**

```
(FLEXSPI_LUT_OPERAND0 (op0) | FLEXSPI_LUT_NUM_PADS0 (pad0) | FLEXSPI_LUT_OPCODE0 (cmd0) | FLEXSPI_LUT_OPERAND1
 (op1) | \
 FLEXSPI_LUT_NUM_PADS1 (pad1) | FLEXSPI_LUT_OPCODE1 (cmd1))
```

## 25.4 Typedef Documentation

**25.4.1 typedef void(\* flexspi\_transfer\_callback\_t)(FLEXSPI\_Type \*base, flexspi\_handle\_t \*handle, status\_t status, void \*userData)**

## 25.5 Enumeration Type Documentation

### 25.5.1 anonymous enum

Enumerator

- kStatus\_FLEXSPI\_Busy* FLEXSPI is busy.
- kStatus\_FLEXSPI\_SequenceExecutionTimeout* Sequence execution timeout error occurred during FLEXSPI transfer.
- kStatus\_FLEXSPI\_IpCommandSequenceError* IP command Sequence execution timeout error occurred during FLEXSPI transfer.
- kStatus\_FLEXSPI\_IpCommandGrantTimeout* IP command grant timeout error occurred during FLEXSPI transfer.

### 25.5.2 anonymous enum

Enumerator

- kFLEXSPI\_Command\_STOP* Stop execution, deassert CS.
- kFLEXSPI\_Command\_SDR* Transmit Command code to Flash, using SDR mode.
- kFLEXSPI\_Command\_RADDR\_SDR* Transmit Row Address to Flash, using SDR mode.
- kFLEXSPI\_Command\_CADDR\_SDR* Transmit Column Address to Flash, using SDR mode.
- kFLEXSPI\_Command\_MODE1\_SDR* Transmit 1-bit Mode bits to Flash, using SDR mode.
- kFLEXSPI\_Command\_MODE2\_SDR* Transmit 2-bit Mode bits to Flash, using SDR mode.
- kFLEXSPI\_Command\_MODE4\_SDR* Transmit 4-bit Mode bits to Flash, using SDR mode.
- kFLEXSPI\_Command\_MODE8\_SDR* Transmit 8-bit Mode bits to Flash, using SDR mode.
- kFLEXSPI\_Command\_WRITE\_SDR* Transmit Programming Data to Flash, using SDR mode.
- kFLEXSPI\_Command\_READ\_SDR* Receive Read Data from Flash, using SDR mode.

- kFLEXSPI\_Command\_LEARN\_SDR*** Receive Read Data or Preamble bit from Flash, SDR mode.
- kFLEXSPI\_Command\_DATSZ\_SDR*** Transmit Read/Program Data size (byte) to Flash, SDR mode.
- kFLEXSPI\_Command\_DUMMY\_SDR*** Leave data lines undriven by FlexSPI controller.
- kFLEXSPI\_Command\_DUMMY\_RWDS\_SDR*** Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.
- kFLEXSPI\_Command\_DDR*** Transmit Command code to Flash, using DDR mode.
- kFLEXSPI\_Command\_RADDR\_DDR*** Transmit Row Address to Flash, using DDR mode.
- kFLEXSPI\_Command\_CADDR\_DDR*** Transmit Column Address to Flash, using DDR mode.
- kFLEXSPI\_Command\_MODE1\_DDR*** Transmit 1-bit Mode bits to Flash, using DDR mode.
- kFLEXSPI\_Command\_MODE2\_DDR*** Transmit 2-bit Mode bits to Flash, using DDR mode.
- kFLEXSPI\_Command\_MODE4\_DDR*** Transmit 4-bit Mode bits to Flash, using DDR mode.
- kFLEXSPI\_Command\_MODE8\_DDR*** Transmit 8-bit Mode bits to Flash, using DDR mode.
- kFLEXSPI\_Command\_WRITE\_DDR*** Transmit Programming Data to Flash, using DDR mode.
- kFLEXSPI\_Command\_READ\_DDR*** Receive Read Data from Flash, using DDR mode.
- kFLEXSPI\_Command\_LEARN\_DDR*** Receive Read Data or Preamble bit from Flash, DDR mode.
- kFLEXSPI\_Command\_DATSZ\_DDR*** Transmit Read/Program Data size (byte) to Flash, DDR mode.
- kFLEXSPI\_Command\_DUMMY\_DDR*** Leave data lines undriven by FlexSPI controller.
- kFLEXSPI\_Command\_DUMMY\_RWDS\_DDR*** Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.
- kFLEXSPI\_Command\_JUMP\_ON\_CS*** Stop execution, deassert CS and save operand[7:0] as the instruction start pointer for next sequence.

### 25.5.3 enum flexspi\_pad\_t

Enumerator

- kFLEXSPI\_1PAD*** Transmit command/address and transmit/receive data only through DATA0/DATA1.
- kFLEXSPI\_2PAD*** Transmit command/address and transmit/receive data only through DATA[1:0].
- kFLEXSPI\_4PAD*** Transmit command/address and transmit/receive data only through DATA[3:0].
- kFLEXSPI\_8PAD*** Transmit command/address and transmit/receive data only through DATA[7:0].

### 25.5.4 enum flexspi\_flags\_t

Enumerator

- kFLEXSPI\_SequenceExecutionTimeoutFlag*** Sequence execution timeout.

***kFLEXSPI\_AhbBusTimeoutFlag*** AHB Bus timeout.

***kFLEXSPI\_SckStoppedBecauseTxEmptyFlag*** SCK is stopped during command sequence because Async TX FIFO empty.

***kFLEXSPI\_SckStoppedBecauseRxFullFlag*** SCK is stopped during command sequence because Async RX FIFO full.

***kFLEXSPI\_DataLearningFailedFlag*** Data learning failed.

***kFLEXSPI\_IpTxFifoWatermarkEmptyFlag*** IP TX FIFO WaterMark empty.

***kFLEXSPI\_IpRxFifoWatermarkAvailableFlag*** IP RX FIFO WaterMark available.

***kFLEXSPI\_AhbCommandSequenceErrorFlag*** AHB triggered Command Sequences Error.

***kFLEXSPI\_IpCommandSequenceErrorFlag*** IP triggered Command Sequences Error.

***kFLEXSPI\_AhbCommandGrantTimeoutFlag*** AHB triggered Command Sequences Grant Timeout.

***kFLEXSPI\_IpCommandGrantTimeoutFlag*** IP triggered Command Sequences Grant Timeout.

***kFLEXSPI\_IpCommandExecutionDoneFlag*** IP triggered Command Sequences Execution finished.

***kFLEXSPI\_AllInterruptFlags*** All flags.

### 25.5.5 enum flexspi\_read\_sample\_clock\_t

Enumerator

***kFLEXSPI\_ReadSampleClkLoopbackInternally*** Dummy Read strobe generated by FlexSPI Controller and loopback internally.

***kFLEXSPI\_ReadSampleClkLoopbackFromDqsPad*** Dummy Read strobe generated by FlexSPI Controller and loopback from DQS pad.

***kFLEXSPI\_ReadSampleClkLoopbackFromSckPad*** SCK output clock and loopback from SCK pad.

***kFLEXSPI\_ReadSampleClkExternalInputFromDqsPad*** Flash provided Read strobe and input from DQS pad.

### 25.5.6 enum flexspi\_cs\_interval\_cycle\_unit\_t

Enumerator

***kFLEXSPI\_CsIntervalUnit1SckCycle*** Chip selection interval: CSINTERVAL \* 1 serial clock cycle.

***kFLEXSPI\_CsIntervalUnit256SckCycle*** Chip selection interval: CSINTERVAL \* 256 serial clock cycle.

### 25.5.7 enum flexspi\_ahb\_write\_wait\_unit\_t

Enumerator

*kFLEXSPI\_AhbWriteWaitUnit2AhbCycle* AWRWAIT unit is 2 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit8AhbCycle* AWRWAIT unit is 8 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit32AhbCycle* AWRWAIT unit is 32 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit128AhbCycle* AWRWAIT unit is 128 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit512AhbCycle* AWRWAIT unit is 512 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit2048AhbCycle* AWRWAIT unit is 2048 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit8192AhbCycle* AWRWAIT unit is 8192 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit32768AhbCycle* AWRWAIT unit is 32768 ahb clock cycle.

### 25.5.8 enum flexspi\_ip\_error\_code\_t

Enumerator

*kFLEXSPI\_IpCmdErrorNoError* No error.  
*kFLEXSPI\_IpCmdErrorJumpOnCsInIpCmd* IP command with JMP\_ON\_CS instruction used.  
*kFLEXSPI\_IpCmdErrorUnknownOpCode* Unknown instruction opcode in the sequence.  
*kFLEXSPI\_IpCmdErrorSdrDummyInDdrSequence* Instruction DUMMY\_SDR/DUMMY\_RW-DS\_SDR used in DDR sequence.  
*kFLEXSPI\_IpCmdErrorDdrDummyInSdrSequence* Instruction DUMMY\_DDR/DUMMY\_RW-DS\_DDR used in SDR sequence.  
*kFLEXSPI\_IpCmdErrorInvalidAddress* Flash access start address exceed the whole flash address range (A1/A2/B1/B2).  
*kFLEXSPI\_IpCmdErrorSequenceExecutionTimeout* Sequence execution timeout.  
*kFLEXSPI\_IpCmdErrorFlashBoundaryAcrosss* Flash boundary crossed.

### 25.5.9 enum flexspi\_ahb\_error\_code\_t

Enumerator

*kFLEXSPI\_AhbCmdErrorNoError* No error.  
*kFLEXSPI\_AhbCmdErrorJumpOnCsInWriteCmd* AHB Write command with JMP\_ON\_CS instruction used in the sequence.  
*kFLEXSPI\_AhbCmdErrorUnknownOpCode* Unknown instruction opcode in the sequence.  
*kFLEXSPI\_AhbCmdErrorSdrDummyInDdrSequence* Instruction DUMMY\_SDR/DUMMY\_R-WDS\_SDR used in DDR sequence.  
*kFLEXSPI\_AhbCmdErrorDdrDummyInSdrSequence* Instruction DUMMY\_DDR/DUMMY\_R-WDS\_DDR used in SDR sequence.  
*kFLEXSPI\_AhbCmdSequenceExecutionTimeout* Sequence execution timeout.



### 25.5.10 enum flexspi\_port\_t

Enumerator

*kFLEXSPI\_PortA1* Access flash on A1 port.  
*kFLEXSPI\_PortA2* Access flash on A2 port.  
*kFLEXSPI\_PortB1* Access flash on B1 port.  
*kFLEXSPI\_PortB2* Access flash on B2 port.

### 25.5.11 enum flexspi\_arb\_command\_source\_t

### 25.5.12 enum flexspi\_command\_type\_t

Enumerator

*kFLEXSPI\_Command* FlexSPI operation: Only command, both TX and Rx buffer are ignored.  
*kFLEXSPI\_Config* FlexSPI operation: Configure device mode, the TX fifo size is fixed in LUT.

## 25.6 Function Documentation

### 25.6.1 uint32\_t FLEXSPI\_GetInstance ( FLEXSPI\_Type \* *base* )

Parameters

|             |                       |
|-------------|-----------------------|
| <i>base</i> | FLEXSPI base pointer. |
|-------------|-----------------------|

### 25.6.2 status\_t FLEXSPI\_CheckAndClearError ( FLEXSPI\_Type \* *base*, uint32\_t *status* )

Parameters

|               |                       |
|---------------|-----------------------|
| <i>base</i>   | FLEXSPI base pointer. |
| <i>status</i> | interrupt status.     |

### 25.6.3 void FLEXSPI\_Init ( FLEXSPI\_Type \* *base*, const flexspi\_config\_t \* *config* )

This function enables the clock for FLEXSPI and also configures the FLEXSPI with the input configure parameters. Users should call this function before any FLEXSPI operations.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address. |
| <i>config</i> | FLEXSPI configure structure.     |

#### 25.6.4 void FLEXSPI\_GetDefaultConfig ( flexspi\_config\_t \* config )

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>config</i> | FLEXSPI configuration structure. |
|---------------|----------------------------------|

#### 25.6.5 void FLEXSPI\_Deinit ( FLEXSPI\_Type \* base )

Clears the FLEXSPI state and FLEXSPI module registers.

Parameters

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

#### 25.6.6 void FLEXSPI\_UpdateDIIValue ( FLEXSPI\_Type \* base, flexspi\_device\_config\_t \* config, flexspi\_port\_t port )

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address. |
| <i>config</i> | Flash configuration parameters.  |
| <i>port</i>   | FLEXSPI Operation port.          |

#### 25.6.7 void FLEXSPI\_SetFlashConfig ( FLEXSPI\_Type \* base, flexspi\_device\_config\_t \* config, flexspi\_port\_t port )

This function configures the connected device relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the connected device.

## Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address. |
| <i>config</i> | Flash configuration parameters.  |
| <i>port</i>   | FLEXSPI Operation port.          |

### 25.6.8 static void FLEXSPI\_SoftwareReset ( FLEXSPI\_Type \* *base* ) [inline], [static]

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

## Parameters

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

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

## Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                |
| <i>enable</i> | True means enable FLEXSPI, false means disable. |

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

## Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
| <i>mask</i> | FLEXSPI interrupt source.        |

### 25.6.11 static void FLEXSPI\_DisableInterrupts ( FLEXSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
| <i>mask</i> | FLEXSPI interrupt source.        |

**25.6.12** `static void FLEXSPI_EnableTxDMA ( FLEXSPI_Type * base, bool enable )`  
`[inline], [static]`

## Parameters

|               |                                                                                |
|---------------|--------------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                               |
| <i>enable</i> | Enable flag for transmit DMA request. Pass true for enable, false for disable. |

**25.6.13** `static void FLEXSPI_EnableRxDMA ( FLEXSPI_Type * base, bool enable )`  
`[inline], [static]`

## Parameters

|               |                                                                               |
|---------------|-------------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                              |
| <i>enable</i> | Enable flag for receive DMA request. Pass true for enable, false for disable. |

**25.6.14** `static uint32_t FLEXSPI_GetTxFifoAddress ( FLEXSPI_Type * base )`  
`[inline], [static]`

## Parameters

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

## Return values

|            |                  |
|------------|------------------|
| <i>The</i> | tx fifo address. |
|------------|------------------|

**25.6.15** `static uint32_t FLEXSPI_GetRxFifoAddress ( FLEXSPI_Type * base )`  
`[inline], [static]`

Parameters

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

Return values

|            |                  |
|------------|------------------|
| <i>The</i> | rx fifo address. |
|------------|------------------|

**25.6.16 static void FLEXSPI\_ResetFifos ( FLEXSPI\_Type \* *base*, bool *txFifo*, bool *rxFifo* ) [inline], [static]**

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address. |
| <i>txFifo</i> | Pass true to reset TX FIFO.      |
| <i>rxFifo</i> | Pass true to reset RX FIFO.      |

**25.6.17 static void FLEXSPI\_GetFifoCounts ( FLEXSPI\_Type \* *base*, size\_t \* *txCount*, size\_t \* *rxCount* ) [inline], [static]**

Parameters

|     |                |                                                                                                                              |
|-----|----------------|------------------------------------------------------------------------------------------------------------------------------|
|     | <i>base</i>    | FLEXSPI peripheral base address.                                                                                             |
| out | <i>txCount</i> | Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required. |
| out | <i>rxCount</i> | Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.  |

**25.6.18 static uint32\_t FLEXSPI\_GetInterruptStatusFlags ( FLEXSPI\_Type \* *base* ) [inline], [static]**

Parameters

---

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

Return values

|                  |                                                                                                |
|------------------|------------------------------------------------------------------------------------------------|
| <i>interrupt</i> | status flag, use status flag to AND <code>flexspi_flags_t</code> could get the related status. |
|------------------|------------------------------------------------------------------------------------------------|

**25.6.19** `static void FLEXSPI_ClearInterruptStatusFlags ( FLEXSPI_Type * base, uint32_t mask ) [inline], [static]`

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
| <i>mask</i> | FLEXSPI interrupt source.        |

**25.6.20** `static void FLEXSPI_GetDataLearningPhase ( FLEXSPI_Type * base, uint8_t * portAPhase, uint8_t * portBPhase ) [inline], [static]`

Parameters

|                   |                                                                                  |
|-------------------|----------------------------------------------------------------------------------|
| <i>base</i>       | FLEXSPI peripheral base address.                                                 |
| <i>portAPhase</i> | Pointer to a uint8_t type variable to receive the selected clock phase on PORTA. |
| <i>portBPhase</i> | Pointer to a uint8_t type variable to receive the selected clock phase on PORTB. |

**25.6.21** `static flexspi_arb_command_source_t FLEXSPI_GetArbitrator-CommandSource ( FLEXSPI_Type * base ) [inline], [static]`

Parameters

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

Return values

---

|                |                                     |
|----------------|-------------------------------------|
| <i>trigger</i> | source of current command sequence. |
|----------------|-------------------------------------|

**25.6.22** `static flexspi_ip_error_code_t FLEXSPI_GetIPCommandErrorCode ( FLEXSPI_Type * base, uint8_t * index ) [inline], [static]`

Parameters

|              |                                                                                       |
|--------------|---------------------------------------------------------------------------------------|
| <i>base</i>  | FLEXSPI peripheral base address.                                                      |
| <i>index</i> | Pointer to a uint8_t type variable to receive the sequence index when error detected. |

Return values

|              |                                      |
|--------------|--------------------------------------|
| <i>error</i> | code when IP command error detected. |
|--------------|--------------------------------------|

**25.6.23** `static flexspi_ahb_error_code_t FLEXSPI_GetAHBCommandErrorCode ( FLEXSPI_Type * base, uint8_t * index ) [inline], [static]`

Parameters

|              |                                                                                       |
|--------------|---------------------------------------------------------------------------------------|
| <i>base</i>  | FLEXSPI peripheral base address.                                                      |
| <i>index</i> | Pointer to a uint8_t type variable to receive the sequence index when error detected. |

Return values

|              |                                       |
|--------------|---------------------------------------|
| <i>error</i> | code when AHB command error detected. |
|--------------|---------------------------------------|

**25.6.24** `static bool FLEXSPI_GetBusIdleStatus ( FLEXSPI_Type * base ) [inline], [static]`

Parameters

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

Return values

|              |              |
|--------------|--------------|
| <i>true</i>  | Bus is idle. |
| <i>false</i> | Bus is busy. |

**25.6.25 void FLEXSPI\_UpdateRxSampleClock ( FLEXSPI\_Type \* *base*, flexspi\_read\_sample\_clock\_t *clockSource* )**

Parameters

|                    |                                                                 |
|--------------------|-----------------------------------------------------------------|
| <i>base</i>        | FLEXSPI peripheral base address.                                |
| <i>clockSource</i> | clockSource of type <a href="#">flexspi_read_sample_clock_t</a> |

**25.6.26 static void FLEXSPI\_EnableIPParallelMode ( FLEXSPI\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                    |
| <i>enable</i> | True means enable parallel mode, false means disable parallel mode. |

**25.6.27 static void FLEXSPI\_EnableAHBParallelMode ( FLEXSPI\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                    |
| <i>enable</i> | True means enable parallel mode, false means disable parallel mode. |

**25.6.28 void FLEXSPI\_UpdateLUT ( FLEXSPI\_Type \* *base*, uint32\_t *index*, const uint32\_t \* *cmd*, uint32\_t *count* )**



## Parameters

|              |                                                                                                                                                                                                                        |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>  | FLEXSPI peripheral base address.                                                                                                                                                                                       |
| <i>index</i> | From which index start to update. It could be any index of the LUT table, which also allows user to update command content inside a command. Each command consists of up to 8 instructions and occupy 4*32-bit memory. |
| <i>cmd</i>   | Command sequence array.                                                                                                                                                                                                |
| <i>count</i> | Number of sequences.                                                                                                                                                                                                   |

**25.6.29** `static void FLEXSPI_WriteData ( FLEXSPI_Type * base, uint32_t data, uint8_t fifoIndex ) [inline], [static]`

## Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>base</i>      | FLEXSPI peripheral base address |
| <i>data</i>      | The data bytes to send          |
| <i>fifoIndex</i> | Destination fifo index.         |

**25.6.30** `static uint32_t FLEXSPI_ReadData ( FLEXSPI_Type * base, uint8_t fifoIndex ) [inline], [static]`

## Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>base</i>      | FLEXSPI peripheral base address |
| <i>fifoIndex</i> | Source fifo index.              |

## Returns

The data in the FIFO.

**25.6.31** `status_t FLEXSPI_WriteBlocking ( FLEXSPI_Type * base, uint8_t * buffer, size_t size )`

## Note

This function blocks via polling until all bytes have been sent.

## Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address  |
| <i>buffer</i> | The data bytes to send           |
| <i>size</i>   | The number of data bytes to send |

## Return values

|                                                 |                                    |
|-------------------------------------------------|------------------------------------|
| <i>kStatus_Success</i>                          | write success without error        |
| <i>kStatus_FLEXSPI_SequenceExecutionTimeout</i> | sequence execution timeout         |
| <i>kStatus_FLEXSPI_IpCommandSequenceError</i>   | IP command sequence error detected |
| <i>kStatus_FLEXSPI_IpCommandGrantTimeout</i>    | IP command grant timeout detected  |

### 25.6.32 `status_t FLEXSPI_ReadBlocking ( FLEXSPI_Type * base, uint8_t * buffer, size_t size )`

## Note

This function blocks via polling until all bytes have been sent.

## Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address     |
| <i>buffer</i> | The data bytes to send              |
| <i>size</i>   | The number of data bytes to receive |

## Return values

|                                                 |                            |
|-------------------------------------------------|----------------------------|
| <i>kStatus_Success</i>                          | read success without error |
| <i>kStatus_FLEXSPI_SequenceExecutionTimeout</i> | sequence execution timeout |

|                                                |                                     |
|------------------------------------------------|-------------------------------------|
| <i>kStatus_FLEXSPI_Ip-CommandSequenceError</i> | IP command sequencen error detected |
| <i>kStatus_FLEXSPI_Ip-CommandGrantTimeout</i>  | IP command grant timeout detected   |

**25.6.33 status\_t FLEXSPI\_TransferBlocking ( FLEXSPI\_Type \* *base*, flexspi\_transfer\_t \* *xfer* )**

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | FLEXSPI peripheral base address    |
| <i>xfer</i> | pointer to the transfer structure. |

Return values

|                                                   |                                        |
|---------------------------------------------------|----------------------------------------|
| <i>kStatus_Success</i>                            | command transfer success without error |
| <i>kStatus_FLEXSPI_-SequenceExecution-Timeout</i> | sequence execution timeout             |
| <i>kStatus_FLEXSPI_Ip-CommandSequenceError</i>    | IP command sequence error detected     |
| <i>kStatus_FLEXSPI_Ip-CommandGrantTimeout</i>     | IP command grant timeout detected      |

**25.6.34 void FLEXSPI\_TransferCreateHandle ( FLEXSPI\_Type \* *base*, flexspi\_handle\_t \* *handle*, flexspi\_transfer\_callback\_t *callback*, void \* *userData* )**

Parameters

|               |                                                                    |
|---------------|--------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                   |
| <i>handle</i> | pointer to flexspi_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. |

### 25.6.35 `status_t FLEXSPI_TransferNonBlocking ( FLEXSPI_Type * base, flexspi_handle_t * handle, flexspi_transfer_t * xfer )`

#### Note

Calling the API returns immediately after transfer initiates. The user needs to call `FLEXSPI_GetTransferCount` to poll the transfer status to check whether the transfer is finished. If the return status is not `kStatus_FLEXSPI_Busy`, the transfer is finished. For `FLEXSPI_Read`, the `dataSize` should be multiple of rx watermark level, or FLEXSPI could not read data properly.

#### Parameters

|               |                                                                                     |
|---------------|-------------------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                                    |
| <i>handle</i> | pointer to <code>flexspi_handle_t</code> structure which stores the transfer state. |
| <i>xfer</i>   | pointer to <code>flexspi_transfer_t</code> structure.                               |

#### Return values

|                             |                                           |
|-----------------------------|-------------------------------------------|
| <i>kStatus_Success</i>      | Successfully start the data transmission. |
| <i>kStatus_FLEXSPI_Busy</i> | Previous transmission still not finished. |

### 25.6.36 `status_t FLEXSPI_TransferGetCount ( FLEXSPI_Type * base, flexspi_handle_t * handle, size_t * count )`

#### Parameters

|               |                                                                                     |
|---------------|-------------------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                                    |
| <i>handle</i> | pointer to <code>flexspi_handle_t</code> 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. |

### 25.6.37 void FLEXSPI\_TransferAbort ( FLEXSPI\_Type \* *base*, flexspi\_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>   | FLEXSPI peripheral base address.                                      |
| <i>handle</i> | pointer to flexspi_handle_t structure which stores the transfer state |

### 25.6.38 void FLEXSPI\_TransferHandleIRQ ( FLEXSPI\_Type \* *base*, flexspi\_handle\_t \* *handle* )

#### Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.       |
| <i>handle</i> | pointer to flexspi_handle_t structure. |

## 25.7 FLEXSPI eDMA Driver

### 25.7.1 Overview

#### Data Structures

- struct `flexspi_edma_handle_t`  
*FLEXSPI DMA transfer handle, users should not touch the content of the handle. [More...](#)*

#### Typedefs

- typedef void(\* `flexspi_edma_callback_t`)(FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle, `status_t` status, void \*userData)  
*FLEXSPI eDMA transfer callback function for finish and error.*

#### Enumerations

- enum `flexspi_edma_transfer_nsize_t` {  
`kFLEXPSI_EDMAAnSize1Bytes = 0x1U,`  
`kFLEXPSI_EDMAAnSize2Bytes = 0x2U,`  
`kFLEXPSI_EDMAAnSize4Bytes = 0x4U,`  
`kFLEXPSI_EDMAAnSize8Bytes = 0x8U,`  
`kFLEXPSI_EDMAAnSize32Bytes = 0x20U }`  
*eDMA transfer configuration*

#### Driver version

- #define `FSL_FLEXSPI_EDMA_DRIVER_VERSION` (MAKE\_VERSION(2, 3, 3))  
*FLEXSPI EDMA driver version.*

#### FLEXSPI eDMA Transactional

- void `FLEXSPI_TransferCreateHandleEDMA` (FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle, `flexspi_edma_callback_t` callback, void \*userData, `edma_handle_t` \*txDmaHandle, `edma_handle_t` \*rxDmaHandle)  
*Initializes the FLEXSPI handle for transfer which is used in transactional functions and set the callback.*
- void `FLEXSPI_TransferUpdateSizeEDMA` (FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle, `flexspi_edma_transfer_nsize_t` nsize)  
*Update FLEXSPI EDMA transfer source data transfer size(SSIZE) and destination data transfer size(DSIZE).*
- `status_t` `FLEXSPI_TransferEDMA` (FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle, `flexspi_transfer_t` \*xfer)  
*Transfers FLEXSPI data using an eDMA non-blocking method.*

- void [FLEXSPI\\_TransferAbortEDMA](#) (FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle)  
*Aborts the transfer data using eDMA.*
- [status\\_t FLEXSPI\\_TransferGetTransferCountEDMA](#) (FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets the transferred counts of transfer.*

## 25.7.2 Data Structure Documentation

### 25.7.2.1 struct flexspi\_edma\_handle

#### Data Fields

- [edma\\_handle\\_t \\* txDmaHandle](#)  
*eDMA handler for FLEXSPI Tx.*
- [edma\\_handle\\_t \\* rxDmaHandle](#)  
*eDMA handler for FLEXSPI Rx.*
- [size\\_t transferSize](#)  
*Bytes need to transfer.*
- [flexspi\\_edma\\_transfer\\_nsize\\_t nsize](#)  
*eDMA SSIZE/DSIZE in each transfer.*
- [uint8\\_t nbytes](#)  
*eDMA minor byte transfer count initially configured.*
- [uint8\\_t count](#)  
*The transfer data count in a DMA request.*
- [uint32\\_t state](#)  
*Internal state for FLEXSPI eDMA transfer.*
- [flexspi\\_edma\\_callback\\_t completionCallback](#)  
*A callback function called after the eDMA transfer is finished.*
- [void \\* userData](#)  
*User callback parameter.*

## Field Documentation

- (1) `edma_handle_t* flexspi_edma_handle_t::txDmaHandle`
- (2) `edma_handle_t* flexspi_edma_handle_t::rxDmaHandle`
- (3) `size_t flexspi_edma_handle_t::transferSize`
- (4) `flexspi_edma_transfer_nsize_t flexspi_edma_handle_t::nsize`
- (5) `uint8_t flexspi_edma_handle_t::nbytes`
- (6) `uint8_t flexspi_edma_handle_t::count`
- (7) `uint32_t flexspi_edma_handle_t::state`
- (8) `flexspi_edma_callback_t flexspi_edma_handle_t::completionCallback`

## 25.7.3 Macro Definition Documentation

25.7.3.1 `#define FSL_FLEXSPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 3, 3))`

## 25.7.4 Enumeration Type Documentation

25.7.4.1 `enum flexspi_edma_transfer_nsize_t`

Enumerator

- kFLEXPSI\_EDMAnSize1Bytes* Source/Destination data transfer size is 1 byte every time.
- kFLEXPSI\_EDMAnSize2Bytes* Source/Destination data transfer size is 2 bytes every time.
- kFLEXPSI\_EDMAnSize4Bytes* Source/Destination data transfer size is 4 bytes every time.
- kFLEXPSI\_EDMAnSize8Bytes* Source/Destination data transfer size is 8 bytes every time.
- kFLEXPSI\_EDMAnSize32Bytes* Source/Destination data transfer size is 32 bytes every time.

## 25.7.5 Function Documentation

25.7.5.1 `void FLEXSPI_TransferCreateHandleEDMA ( FLEXSPI_Type * base,  
flexspi_edma_handle_t * handle, flexspi_edma_callback_t callback, void *  
userData, edma_handle_t * txDmaHandle, edma_handle_t * rxDmaHandle )`

Parameters

---



|                    |                                                |
|--------------------|------------------------------------------------|
| <i>base</i>        | FLEXSPI peripheral base address                |
| <i>handle</i>      | Pointer to flexspi_edma_handle_t structure     |
| <i>callback</i>    | FLEXSPI callback, NULL means no callback.      |
| <i>userData</i>    | User callback function data.                   |
| <i>txDmaHandle</i> | User requested DMA handle for TX DMA transfer. |
| <i>rxDmaHandle</i> | User requested DMA handle for RX DMA transfer. |

### 25.7.5.2 void FLEXSPI\_TransferUpdateSizeEDMA ( FLEXSPI\_Type \* *base*, flexspi\_edma\_handle\_t \* *handle*, flexspi\_edma\_transfer\_nsize\_t *nsize* )

Parameters

|               |                                                                                                                   |
|---------------|-------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address                                                                                   |
| <i>handle</i> | Pointer to flexspi_edma_handle_t structure                                                                        |
| <i>nsize</i>  | FLEXSPI DMA transfer data transfer size(SSIZE/DSIZE), by default the size is kFLEXPSI_EDMAAnSize1Bytes(one byte). |

See Also

[flexspi\\_edma\\_transfer\\_nsize\\_t](#) .

### 25.7.5.3 status\_t FLEXSPI\_TransferEDMA ( FLEXSPI\_Type \* *base*, flexspi\_edma\_handle\_t \* *handle*, flexspi\_transfer\_t \* *xfer* )

This function writes/receives data to/from the FLEXSPI transmit/receive FIFO. This function is non-blocking.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.           |
| <i>handle</i> | Pointer to flexspi_edma_handle_t structure |
| <i>xfer</i>   | FLEXSPI transfer structure.                |

Return values

---

|                                |                                                                                                              |
|--------------------------------|--------------------------------------------------------------------------------------------------------------|
| <i>kStatus_FLEXSPI_Busy</i>    | FLEXSPI is busy transfer.                                                                                    |
| <i>kStatus_InvalidArgument</i> | The watermark configuration is invalid, the watermark should be power of 2 to do successfully EDMA transfer. |
| <i>kStatus_Success</i>         | FLEXSPI successfully start edma transfer.                                                                    |

**25.7.5.4 void FLEXSPI\_TransferAbortEDMA ( FLEXSPI\_Type \* *base*, flexspi\_edma\_handle\_t \* *handle* )**

This function aborts the transfer data using eDMA.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.           |
| <i>handle</i> | Pointer to flexspi_edma_handle_t structure |

**25.7.5.5 status\_t FLEXSPI\_TransferGetTransferCountEDMA ( FLEXSPI\_Type \* *base*, flexspi\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

|               |                                             |
|---------------|---------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.            |
| <i>handle</i> | Pointer to flexspi_edma_handle_t structure. |
| <i>count</i>  | Bytes transfer.                             |

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. |

## Chapter 26

# GPC: General Power Controller Driver

### 26.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General Power Controller (GPC) module of MCUXpresso SDK devices.

API functions are provided to configure the system about working in dedicated power mode. There are mainly about enabling the power for memory, enabling the wakeup sources for STOP modes, and power up/down operations for various peripherals.

### Functions

- void [GPC\\_EnableIRQ](#) (GPC\_Type \*base, uint32\_t irqId)  
*Enable the IRQ.*
- void [GPC\\_DisableIRQ](#) (GPC\_Type \*base, uint32\_t irqId)  
*Disable the IRQ.*
- bool [GPC\\_GetIRQStatusFlag](#) (GPC\_Type \*base, uint32\_t irqId)  
*Get the IRQ/Event flag.*
- static void [GPC\\_RequestMEGAPowerOn](#) (GPC\_Type \*base, bool enable)  
*Requests the MEGA power switch sequence.*

### Driver version

- #define [FSL\\_GPC\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 1))  
*GPC driver version 2.1.1.*

### 26.2 Macro Definition Documentation

#### 26.2.1 #define FSL\_GPC\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))

### 26.3 Function Documentation

#### 26.3.1 void GPC\_EnableIRQ ( GPC\_Type \* base, uint32\_t irqId )

Parameters

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

|              |                                                                                                 |
|--------------|-------------------------------------------------------------------------------------------------|
| <i>irqId</i> | ID number of IRQ to be enabled, available range is 32-159. 0-31 is available in some platforms. |
|--------------|-------------------------------------------------------------------------------------------------|

### 26.3.2 void GPC\_DisableIRQ ( GPC\_Type \* *base*, uint32\_t *irqId* )

Parameters

|              |                                                                                                  |
|--------------|--------------------------------------------------------------------------------------------------|
| <i>base</i>  | GPC peripheral base address.                                                                     |
| <i>irqId</i> | ID number of IRQ to be disabled, available range is 32-159. 0-31 is available in some platforms. |

### 26.3.3 bool GPC\_GetIRQStatusFlag ( GPC\_Type \* *base*, uint32\_t *irqId* )

Parameters

|              |                                                                                                 |
|--------------|-------------------------------------------------------------------------------------------------|
| <i>base</i>  | GPC peripheral base address.                                                                    |
| <i>irqId</i> | ID number of IRQ to be enabled, available range is 32-159. 0-31 is available in some platforms. |

Returns

Indicated IRQ/Event is asserted or not.

### 26.3.4 static void GPC\_RequestMEGAPowerOn ( GPC\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                                           |
|---------------|-----------------------------------------------------------|
| <i>base</i>   | GPC peripheral base address.                              |
| <i>enable</i> | Enable the power on sequence, or the power down sequence. |

## Chapter 27

# GPT: General Purpose Timer

### 27.1 Overview

The MCUXpresso SDK provides a driver for the General Purpose Timer (GPT) of MCUXpresso SDK devices.

### 27.2 Function groups

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

#### 27.2.1 Initialization and deinitialization

The function `GPT_Init()` initializes the gpt with specified configurations. The function `GPT_GetDefaultConfig()` gets the default configurations. The initialization function configures the restart/free-run mode and input selection when running.

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

### 27.3 Typical use case

#### 27.3.1 GPT interrupt example

Set up a channel to trigger a periodic interrupt after every 1 second. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/gpt`

### Data Structures

- struct `gpt_config_t`  
*Structure to configure the running mode. [More...](#)*

### Enumerations

- enum `gpt_clock_source_t` {  
    `kGPT_ClockSource_Off` = 0U,  
    `kGPT_ClockSource_Periph` = 1U,  
    `kGPT_ClockSource_HighFreq` = 2U,  
    `kGPT_ClockSource_Ext` = 3U,  
    `kGPT_ClockSource_LowFreq` = 4U,  
    `kGPT_ClockSource_Osc` = 5U }  
*List of clock sources.*

- enum `gpt_input_capture_channel_t` {  
`kGPT_InputCapture_Channel1` = 0U,  
`kGPT_InputCapture_Channel2` = 1U }  
*List of input capture channel number.*
- enum `gpt_input_operation_mode_t` {  
`kGPT_InputOperation_Disabled` = 0U,  
`kGPT_InputOperation_RiseEdge` = 1U,  
`kGPT_InputOperation_FallEdge` = 2U,  
`kGPT_InputOperation_BothEdge` = 3U }  
*List of input capture operation mode.*
- enum `gpt_output_compare_channel_t` {  
`kGPT_OutputCompare_Channel1` = 0U,  
`kGPT_OutputCompare_Channel2` = 1U,  
`kGPT_OutputCompare_Channel3` = 2U }  
*List of output compare channel number.*
- enum `gpt_output_operation_mode_t` {  
`kGPT_OutputOperation_Disconnected` = 0U,  
`kGPT_OutputOperation_Toggle` = 1U,  
`kGPT_OutputOperation_Clear` = 2U,  
`kGPT_OutputOperation_Set` = 3U,  
`kGPT_OutputOperation_Activelow` = 4U }  
*List of output compare operation mode.*
- enum `gpt_interrupt_enable_t` {  
`kGPT_OutputCompare1InterruptEnable` = GPT\_IR\_OF1IE\_MASK,  
`kGPT_OutputCompare2InterruptEnable` = GPT\_IR\_OF2IE\_MASK,  
`kGPT_OutputCompare3InterruptEnable` = GPT\_IR\_OF3IE\_MASK,  
`kGPT_InputCapture1InterruptEnable` = GPT\_IR\_IF1IE\_MASK,  
`kGPT_InputCapture2InterruptEnable` = GPT\_IR\_IF2IE\_MASK,  
`kGPT_RollOverFlagInterruptEnable` = GPT\_IR\_ROVIE\_MASK }  
*List of GPT interrupts.*
- enum `gpt_status_flag_t` {  
`kGPT_OutputCompare1Flag` = GPT\_SR\_OF1\_MASK,  
`kGPT_OutputCompare2Flag` = GPT\_SR\_OF2\_MASK,  
`kGPT_OutputCompare3Flag` = GPT\_SR\_OF3\_MASK,  
`kGPT_InputCapture1Flag` = GPT\_SR\_IF1\_MASK,  
`kGPT_InputCapture2Flag` = GPT\_SR\_IF2\_MASK,  
`kGPT_RollOverFlag` = GPT\_SR\_ROV\_MASK }  
*Status flag.*

## Driver version

- #define `FSL_GPT_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 4)`)

## Initialization and deinitialization

- void `GPT_Init` (`GPT_Type *base`, const `gpt_config_t *initConfig`)  
*Initialize GPT to reset state and initialize running mode.*

- void [GPT\\_Deinit](#) (GPT\_Type \*base)  
*Disables the module and gates the GPT clock.*
- void [GPT\\_GetDefaultConfig](#) ([gpt\\_config\\_t](#) \*config)  
*Fills in the GPT configuration structure with default settings.*

## Software Reset

- static void [GPT\\_SoftwareReset](#) (GPT\_Type \*base)  
*Software reset of GPT module.*

## Clock source and frequency control

- static void [GPT\\_SetClockSource](#) (GPT\_Type \*base, [gpt\\_clock\\_source\\_t](#) gptClkSource)  
*Set clock source of GPT.*
- static [gpt\\_clock\\_source\\_t](#) [GPT\\_GetClockSource](#) (GPT\_Type \*base)  
*Get clock source of GPT.*
- static void [GPT\\_SetClockDivider](#) (GPT\_Type \*base, uint32\_t divider)  
*Set pre scaler of GPT.*
- static uint32\_t [GPT\\_GetClockDivider](#) (GPT\_Type \*base)  
*Get clock divider in GPT module.*
- static void [GPT\\_SetOscClockDivider](#) (GPT\_Type \*base, uint32\_t divider)  
*OSC 24M pre-scaler before selected by clock source.*
- static uint32\_t [GPT\\_GetOscClockDivider](#) (GPT\_Type \*base)  
*Get OSC 24M clock divider in GPT module.*

## Timer Start and Stop

- static void [GPT\\_StartTimer](#) (GPT\_Type \*base)  
*Start GPT timer.*
- static void [GPT\\_StopTimer](#) (GPT\_Type \*base)  
*Stop GPT timer.*

## Read the timer period

- static uint32\_t [GPT\\_GetCurrentTimerCount](#) (GPT\_Type \*base)  
*Reads the current GPT counting value.*

## GPT Input/Output Signal Control

- static void [GPT\\_SetInputOperationMode](#) (GPT\_Type \*base, [gpt\\_input\\_capture\\_channel\\_t](#) channel, [gpt\\_input\\_operation\\_mode\\_t](#) mode)  
*Set GPT operation mode of input capture channel.*
- static [gpt\\_input\\_operation\\_mode\\_t](#) [GPT\\_GetInputOperationMode](#) (GPT\_Type \*base, [gpt\\_input\\_capture\\_channel\\_t](#) channel)  
*Get GPT operation mode of input capture channel.*
- static uint32\_t [GPT\\_GetInputCaptureValue](#) (GPT\_Type \*base, [gpt\\_input\\_capture\\_channel\\_t](#) channel)  
*Get GPT input capture value of certain channel.*
- static void [GPT\\_SetOutputOperationMode](#) (GPT\_Type \*base, [gpt\\_output\\_compare\\_channel\\_t](#) channel, [gpt\\_output\\_operation\\_mode\\_t](#) mode)

- *Set GPT operation mode of output compare channel.*  
static [gpt\\_output\\_operation\\_mode\\_t](#) [GPT\\_GetOutputOperationMode](#) (GPT\_Type \*base, [gpt\\_output\\_compare\\_channel\\_t](#) channel)
- *Get GPT operation mode of output compare channel.*  
static void [GPT\\_SetOutputCompareValue](#) (GPT\_Type \*base, [gpt\\_output\\_compare\\_channel\\_t](#) channel, uint32\_t value)
- *Set GPT output compare value of output compare channel.*  
static uint32\_t [GPT\\_GetOutputCompareValue](#) (GPT\_Type \*base, [gpt\\_output\\_compare\\_channel\\_t](#) channel)
- *Get GPT output compare value of output compare channel.*  
static void [GPT\\_ForceOutput](#) (GPT\_Type \*base, [gpt\\_output\\_compare\\_channel\\_t](#) channel)  
*Force GPT output action on output compare channel, ignoring comparator.*

## GPT Interrupt and Status Interface

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

## Status Interface

- static uint32\_t [GPT\\_GetStatusFlags](#) (GPT\_Type \*base, [gpt\\_status\\_flag\\_t](#) flags)  
*Get GPT status flags.*
- static void [GPT\\_ClearStatusFlags](#) (GPT\_Type \*base, [gpt\\_status\\_flag\\_t](#) flags)  
*Clears the GPT status flags.*

## 27.4 Data Structure Documentation

### 27.4.1 struct gpt\_config\_t

#### Data Fields

- [gpt\\_clock\\_source\\_t](#) clockSource  
*clock source for GPT module.*
- uint32\_t divider  
*clock divider (prescaler+1) from clock source to counter.*
- bool [enableFreeRun](#)  
*true: FreeRun mode, false: Restart mode.*
- bool [enableRunInWait](#)  
*GPT enabled in wait mode.*
- bool [enableRunInStop](#)  
*GPT enabled in stop mode.*
- bool [enableRunInDoze](#)  
*GPT enabled in doze mode.*
- bool [enableRunInDbg](#)  
*GPT enabled in debug mode.*



- bool `enableMode`  
*true:* counter reset to 0 when enabled;  
*false:* counter retain its value when enabled.

### Field Documentation

- (1) `gpt_clock_source_t gpt_config_t::clockSource`
- (2) `uint32_t gpt_config_t::divider`
- (3) `bool gpt_config_t::enableFreeRun`
- (4) `bool gpt_config_t::enableRunInWait`
- (5) `bool gpt_config_t::enableRunInStop`
- (6) `bool gpt_config_t::enableRunInDoze`
- (7) `bool gpt_config_t::enableRunInDbg`
- (8) `bool gpt_config_t::enableMode`

## 27.5 Enumeration Type Documentation

### 27.5.1 enum `gpt_clock_source_t`

Note

Actual number of clock sources is SoC dependent

Enumerator

*kGPT\_ClockSource\_Off* GPT Clock Source Off.  
*kGPT\_ClockSource\_Periph* GPT Clock Source from Peripheral Clock.  
*kGPT\_ClockSource\_HighFreq* GPT Clock Source from High Frequency Reference Clock.  
*kGPT\_ClockSource\_Ext* GPT Clock Source from external pin.  
*kGPT\_ClockSource\_LowFreq* GPT Clock Source from Low Frequency Reference Clock.  
*kGPT\_ClockSource\_Osc* GPT Clock Source from Crystal oscillator.

### 27.5.2 enum `gpt_input_capture_channel_t`

Enumerator

*kGPT\_InputCapture\_Channel1* GPT Input Capture Channel1.  
*kGPT\_InputCapture\_Channel2* GPT Input Capture Channel2.

### 27.5.3 enum gpt\_input\_operation\_mode\_t

Enumerator

*kGPT\_InputOperation\_Disabled* Don't capture.  
*kGPT\_InputOperation\_RiseEdge* Capture on rising edge of input pin.  
*kGPT\_InputOperation\_FallEdge* Capture on falling edge of input pin.  
*kGPT\_InputOperation\_BothEdge* Capture on both edges of input pin.

### 27.5.4 enum gpt\_output\_compare\_channel\_t

Enumerator

*kGPT\_OutputCompare\_Channel1* Output Compare Channel1.  
*kGPT\_OutputCompare\_Channel2* Output Compare Channel2.  
*kGPT\_OutputCompare\_Channel3* Output Compare Channel3.

### 27.5.5 enum gpt\_output\_operation\_mode\_t

Enumerator

*kGPT\_OutputOperation\_Disconnected* Don't change output pin.  
*kGPT\_OutputOperation\_Toggle* Toggle output pin.  
*kGPT\_OutputOperation\_Clear* Set output pin low.  
*kGPT\_OutputOperation\_Set* Set output pin high.  
*kGPT\_OutputOperation\_Activelow* Generate a active low pulse on output pin.

### 27.5.6 enum gpt\_interrupt\_enable\_t

Enumerator

*kGPT\_OutputCompare1InterruptEnable* Output Compare Channel1 interrupt enable.  
*kGPT\_OutputCompare2InterruptEnable* Output Compare Channel2 interrupt enable.  
*kGPT\_OutputCompare3InterruptEnable* Output Compare Channel3 interrupt enable.  
*kGPT\_InputCapture1InterruptEnable* Input Capture Channel1 interrupt enable.  
*kGPT\_InputCapture2InterruptEnable* Input Capture Channel1 interrupt enable.  
*kGPT\_RollOverFlagInterruptEnable* Counter rolled over interrupt enable.

## 27.5.7 enum gpt\_status\_flag\_t

Enumerator

***kGPT\_OutputCompare1Flag*** Output compare channel 1 event.  
***kGPT\_OutputCompare2Flag*** Output compare channel 2 event.  
***kGPT\_OutputCompare3Flag*** Output compare channel 3 event.  
***kGPT\_InputCapture1Flag*** Input Capture channel 1 event.  
***kGPT\_InputCapture2Flag*** Input Capture channel 2 event.  
***kGPT\_RollOverFlag*** Counter reaches maximum value and rolled over to 0 event.

## 27.6 Function Documentation

### 27.6.1 void GPT\_Init ( GPT\_Type \* *base*, const gpt\_config\_t \* *initConfig* )

Parameters

|                   |                                 |
|-------------------|---------------------------------|
| <i>base</i>       | GPT peripheral base address.    |
| <i>initConfig</i> | GPT mode setting configuration. |

### 27.6.2 void GPT\_Deinit ( GPT\_Type \* *base* )

Parameters

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

### 27.6.3 void GPT\_GetDefaultConfig ( gpt\_config\_t \* *config* )

The default values are:

```
* config->clockSource = kGPT_ClockSource_Periph;
* config->divider = 1U;
* config->enableRunInStop = true;
* config->enableRunInWait = true;
* config->enableRunInDoze = false;
* config->enableRunInDbg = false;
* config->enableFreeRun = false;
* config->enableMode = true;
*
```

Parameters

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

**27.6.4 static void GPT\_SoftwareReset ( GPT\_Type \* *base* ) [inline], [static]**

Parameters

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

**27.6.5 static void GPT\_SetClockSource ( GPT\_Type \* *base*, gpt\_clock\_source\_t *gptClkSource* ) [inline], [static]**

Parameters

|                     |                                                                            |
|---------------------|----------------------------------------------------------------------------|
| <i>base</i>         | GPT peripheral base address.                                               |
| <i>gptClkSource</i> | Clock source (see <a href="#">gpt_clock_source_t</a> typedef enumeration). |

**27.6.6 static gpt\_clock\_source\_t GPT\_GetClockSource ( GPT\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

clock source (see [gpt\\_clock\\_source\\_t](#) typedef enumeration).

**27.6.7 static void GPT\_SetClockDivider ( GPT\_Type \* *base*, uint32\_t *divider* ) [inline], [static]**

Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | GPT peripheral base address. |
| <i>divider</i> | Divider of GPT (1-4096).     |

**27.6.8** `static uint32_t GPT_GetClockDivider ( GPT_Type * base ) [inline], [static]`

Parameters

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

Returns

clock divider in GPT module (1-4096).

**27.6.9** `static void GPT_SetOscClockDivider ( GPT_Type * base, uint32_t divider ) [inline], [static]`

Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | GPT peripheral base address. |
| <i>divider</i> | OSC Divider(1-16).           |

**27.6.10** `static uint32_t GPT_GetOscClockDivider ( GPT_Type * base ) [inline], [static]`

Parameters

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

Returns

OSC clock divider in GPT module (1-16).

**27.6.11** `static void GPT_StartTimer ( GPT_Type * base ) [inline], [static]`

Parameters

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

**27.6.12 static void GPT\_StopTimer ( GPT\_Type \* *base* ) [inline], [static]**

Parameters

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

**27.6.13 static uint32\_t GPT\_GetCurrentTimerCount ( GPT\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

Current GPT counter value.

**27.6.14 static void GPT\_SetInputOperationMode ( GPT\_Type \* *base*,  
gpt\_input\_capture\_channel\_t *channel*, gpt\_input\_operation\_mode\_t *mode*  
) [inline], [static]**

Parameters

|                |                                                                                                        |
|----------------|--------------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                           |
| <i>channel</i> | GPT capture channel (see <a href="#">gpt_input_capture_channel_t</a> typedef enumeration).             |
| <i>mode</i>    | GPT input capture operation mode (see <a href="#">gpt_input_operation_mode_t</a> typedef enumeration). |

**27.6.15 static gpt\_input\_operation\_mode\_t GPT\_GetInputOperationMode ( GPT\_Type \* *base*, gpt\_input\_capture\_channel\_t *channel* ) [inline], [static]**

## Parameters

|                |                                                                                            |
|----------------|--------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                               |
| <i>channel</i> | GPT capture channel (see <a href="#">gpt_input_capture_channel_t</a> typedef enumeration). |

## Returns

GPT input capture operation mode (see [gpt\\_input\\_operation\\_mode\\_t](#) typedef enumeration).

**27.6.16** `static uint32_t GPT_GetInputCaptureValue ( GPT_Type * base,  
gpt_input_capture_channel_t channel ) [inline], [static]`

## Parameters

|                |                                                                                            |
|----------------|--------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                               |
| <i>channel</i> | GPT capture channel (see <a href="#">gpt_input_capture_channel_t</a> typedef enumeration). |

## Returns

GPT input capture value.

**27.6.17** `static void GPT_SetOutputOperationMode ( GPT_Type * base,  
gpt_output_compare_channel_t channel, gpt_output_operation_mode_t  
mode ) [inline], [static]`

## Parameters

|                |                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                       |
| <i>channel</i> | GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration). |
| <i>mode</i>    | GPT output operation mode (see <a href="#">gpt_output_operation_mode_t</a> typedef enumeration).   |

**27.6.18** `static gpt_output_operation_mode_t GPT_GetOutputOperationMode (  
GPT_Type * base, gpt_output_compare_channel_t channel ) [inline],  
[static]`

## Parameters

|                |                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                       |
| <i>channel</i> | GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration). |

## Returns

GPT output operation mode (see [gpt\\_output\\_operation\\_mode\\_t](#) typedef enumeration).

**27.6.19** `static void GPT_SetOutputCompareValue ( GPT_Type * base,  
gpt_output_compare_channel_t channel, uint32_t value ) [inline],  
[static]`

## Parameters

|                |                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                       |
| <i>channel</i> | GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration). |
| <i>value</i>   | GPT output compare value.                                                                          |

**27.6.20** `static uint32_t GPT_GetOutputCompareValue ( GPT_Type * base,  
gpt_output_compare_channel_t channel ) [inline], [static]`

## Parameters

|                |                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                       |
| <i>channel</i> | GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration). |

## Returns

GPT output compare value.

**27.6.21** `static void GPT_ForceOutput ( GPT_Type * base, gpt_output_compare_  
channel_t channel ) [inline], [static]`



## Parameters

|                |                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                       |
| <i>channel</i> | GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration). |

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

## Parameters

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

### 27.6.23 static void GPT\_DisableInterrupts ( GPT\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | GPT peripheral base address                                                                                          |
| <i>mask</i> | The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">gpt_interrupt_enable_t</a> |

### 27.6.24 static uint32\_t GPT\_GetEnabledInterrupts ( GPT\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

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

### 27.6.25 static uint32\_t GPT\_GetStatusFlags ( GPT\_Type \* *base*, gpt\_status\_flag\_t *flags* ) [inline], [static]

## Parameters

|              |                                                                                  |
|--------------|----------------------------------------------------------------------------------|
| <i>base</i>  | GPT peripheral base address.                                                     |
| <i>flags</i> | GPT status flag mask (see <a href="#">gpt_status_flag_t</a> for bit definition). |

## Returns

GPT status, each bit represents one status flag.

**27.6.26** `static void GPT_ClearStatusFlags ( GPT_Type * base, gpt_status_flag_t flags ) [inline], [static]`

## Parameters

|              |                                                                                  |
|--------------|----------------------------------------------------------------------------------|
| <i>base</i>  | GPT peripheral base address.                                                     |
| <i>flags</i> | GPT status flag mask (see <a href="#">gpt_status_flag_t</a> for bit definition). |

## Chapter 28

# GPIO: General-Purpose Input/Output Driver

### 28.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General-Purpose Input/Output (GPIO) module of MCUXpresso SDK devices.

### 28.2 Typical use case

#### 28.2.1 Input Operation

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

### Data Structures

- struct `gpio_pin_config_t`  
*GPIO Init structure definition. [More...](#)*

### Enumerations

- enum `gpio_pin_direction_t` {  
    `kGPIO_DigitalInput` = 0U,  
    `kGPIO_DigitalOutput` = 1U }  
*GPIO direction definition.*
- enum `gpio_interrupt_mode_t` {  
    `kGPIO_NoIntmode` = 0U,  
    `kGPIO_IntLowLevel` = 1U,  
    `kGPIO_IntHighLevel` = 2U,  
    `kGPIO_IntRisingEdge` = 3U,  
    `kGPIO_IntFallingEdge` = 4U,  
    `kGPIO_IntRisingOrFallingEdge` = 5U }  
*GPIO interrupt mode definition.*

### Driver version

- #define `FSL_GPIO_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 6)`)  
*GPIO driver version.*

### GPIO Initialization and Configuration functions

- void `GPIO_PinInit` (`GPIO_Type *base`, `uint32_t pin`, const `gpio_pin_config_t *Config`)  
*Initializes the GPIO peripheral according to the specified parameters in the `initConfig`.*

## GPIO Reads and Write Functions

- void [GPIO\\_PinWrite](#) (GPIO\_Type \*base, uint32\_t pin, uint8\_t output)  
*Sets the output level of the individual GPIO pin to logic 1 or 0.*
- static void [GPIO\\_WritePinOutput](#) (GPIO\_Type \*base, uint32\_t pin, uint8\_t output)  
*Sets the output level of the individual GPIO pin to 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\\_SetPinsOutput](#) (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\\_ClearPinsOutput](#) (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.*
- static uint32\_t [GPIO\\_PinRead](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current input value of the GPIO port.*
- static uint32\_t [GPIO\\_ReadPinInput](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current input value of the GPIO port.*

## GPIO Reads Pad Status Functions

- static uint8\_t [GPIO\\_PinReadPadStatus](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current GPIO pin pad status.*
- static uint8\_t [GPIO\\_ReadPadStatus](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current GPIO pin pad status.*

## Interrupts and flags management functions

- void [GPIO\\_PinSetInterruptConfig](#) (GPIO\_Type \*base, uint32\_t pin, [gpio\\_interrupt\\_mode\\_t](#) pinInterruptMode)  
*Sets the current pin interrupt mode.*
- static void [GPIO\\_SetPinInterruptConfig](#) (GPIO\_Type \*base, uint32\_t pin, [gpio\\_interrupt\\_mode\\_t](#) pinInterruptMode)  
*Sets the current pin interrupt mode.*
- static void [GPIO\\_PortEnableInterrupts](#) (GPIO\_Type \*base, uint32\_t mask)  
*Enables the specific pin interrupt.*
- static void [GPIO\\_EnableInterrupts](#) (GPIO\_Type \*base, uint32\_t mask)  
*Enables the specific pin interrupt.*
- static void [GPIO\\_PortDisableInterrupts](#) (GPIO\_Type \*base, uint32\_t mask)  
*Disables the specific pin interrupt.*
- static void [GPIO\\_DisableInterrupts](#) (GPIO\_Type \*base, uint32\_t mask)  
*Disables the specific pin interrupt.*
- static uint32\_t [GPIO\\_PortGetInterruptFlags](#) (GPIO\_Type \*base)  
*Reads individual pin interrupt status.*
- static uint32\_t [GPIO\\_GetPinsInterruptFlags](#) (GPIO\_Type \*base)  
*Reads individual pin interrupt status.*
- static void [GPIO\\_PortClearInterruptFlags](#) (GPIO\_Type \*base, uint32\_t mask)  
*Clears pin interrupt flag.*
- static void [GPIO\\_ClearPinsInterruptFlags](#) (GPIO\_Type \*base, uint32\_t mask)

*Clears pin interrupt flag.*

## 28.3 Data Structure Documentation

### 28.3.1 struct gpio\_pin\_config\_t

#### Data Fields

- [gpio\\_pin\\_direction\\_t direction](#)  
*Specifies the pin direction.*
- uint8\_t [outputLogic](#)  
*Set a default output logic, which has no use in input.*
- [gpio\\_interrupt\\_mode\\_t interruptMode](#)  
*Specifies the pin interrupt mode, a value of [gpio\\_interrupt\\_mode\\_t](#).*

#### Field Documentation

(1) `gpio_pin_direction_t gpio_pin_config_t::direction`

(2) `gpio_interrupt_mode_t gpio_pin_config_t::interruptMode`

## 28.4 Macro Definition Documentation

28.4.1 `#define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 6))`

## 28.5 Enumeration Type Documentation

### 28.5.1 enum gpio\_pin\_direction\_t

Enumerator

*kGPIO\_DigitalInput* Set current pin as digital input.

*kGPIO\_DigitalOutput* Set current pin as digital output.

### 28.5.2 enum gpio\_interrupt\_mode\_t

Enumerator

*kGPIO\_NoIntmode* Set current pin general IO functionality.

*kGPIO\_IntLowLevel* Set current pin interrupt is low-level sensitive.

*kGPIO\_IntHighLevel* Set current pin interrupt is high-level sensitive.

*kGPIO\_IntRisingEdge* Set current pin interrupt is rising-edge sensitive.

*kGPIO\_IntFallingEdge* Set current pin interrupt is falling-edge sensitive.

*kGPIO\_IntRisingOrFallingEdge* Enable the edge select bit to override the ICR register's configuration.

## 28.6 Function Documentation

28.6.1 void GPIO\_PinInit ( GPIO\_Type \* *base*, uint32\_t *pin*, const gpio\_pin\_config\_t \* *Config* )

Parameters

|               |                                                                                                       |
|---------------|-------------------------------------------------------------------------------------------------------|
| <i>base</i>   | GPIO base pointer.                                                                                    |
| <i>pin</i>    | Specifies the pin number                                                                              |
| <i>Config</i> | pointer to a <a href="#">gpio_pin_config_t</a> structure that contains the configuration information. |

### 28.6.2 void GPIO\_PinWrite ( GPIO\_Type \* *base*, uint32\_t *pin*, uint8\_t *output* )

Parameters

|               |                                                                                                                                                                                        |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | GPIO base pointer.                                                                                                                                                                     |
| <i>pin</i>    | GPIO port 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> |

### 28.6.3 static void GPIO\_WritePinOutput ( GPIO\_Type \* *base*, uint32\_t *pin*, uint8\_t *output* ) [inline], [static]

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PinWrite](#).

### 28.6.4 static void GPIO\_PortSet ( GPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.) |
| <i>mask</i> | GPIO pin number macro                                          |

### 28.6.5 static void GPIO\_SetPinsOutput ( GPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PortSet](#).

28.6.6 `static void GPIO_PortClear ( GPIO_Type * base, uint32_t mask )`  
`[inline], [static]`



Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.) |
| <i>mask</i> | GPIO pin number macro                                          |

**28.6.7 static void GPIO\_ClearPinsOutput ( GPIO\_Type \* *base*, uint32\_t *mask* )**  
**[inline], [static]**

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PortClear](#).

**28.6.8 static void GPIO\_PortToggle ( GPIO\_Type \* *base*, uint32\_t *mask* )**  
**[inline], [static]**

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.) |
| <i>mask</i> | GPIO pin number macro                                          |

**28.6.9 static uint32\_t GPIO\_PinRead ( GPIO\_Type \* *base*, uint32\_t *pin* )**  
**[inline], [static]**

Parameters

|             |                       |
|-------------|-----------------------|
| <i>base</i> | GPIO base pointer.    |
| <i>pin</i>  | GPIO port pin number. |

Return values

|             |                   |
|-------------|-------------------|
| <i>GPIO</i> | port input value. |
|-------------|-------------------|

**28.6.10 static uint32\_t GPIO\_ReadPinInput ( GPIO\_Type \* *base*, uint32\_t *pin* )**  
**[inline], [static]**

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PinRead](#).

28.6.11 `static uint8_t GPIO_PinReadPadStatus ( GPIO_Type * base, uint32_t pin )`  
`[inline], [static]`

## Parameters

|             |                       |
|-------------|-----------------------|
| <i>base</i> | GPIO base pointer.    |
| <i>pin</i>  | GPIO port pin number. |

## Return values

|             |                       |
|-------------|-----------------------|
| <i>GPIO</i> | pin pad status value. |
|-------------|-----------------------|

**28.6.12** `static uint8_t GPIO_ReadPadStatus ( GPIO_Type * base, uint32_t pin )  
[inline], [static]`

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PinReadPadStatus](#).

**28.6.13** `void GPIO_PinSetInterruptConfig ( GPIO_Type * base, uint32_t pin,  
gpio_interrupt_mode_t pinInterruptMode )`

## Parameters

|                               |                                                                                                            |
|-------------------------------|------------------------------------------------------------------------------------------------------------|
| <i>base</i>                   | GPIO base pointer.                                                                                         |
| <i>pin</i>                    | GPIO port pin number.                                                                                      |
| <i>pinInterrupt-<br/>Mode</i> | pointer to a <a href="#">gpio_interrupt_mode_t</a> structure that contains the interrupt mode information. |

**28.6.14** `static void GPIO_SetPinInterruptConfig ( GPIO_Type * base, uint32_t pin,  
gpio_interrupt_mode_t pinInterruptMode ) [inline], [static]`

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PinSetInterruptConfig](#).

**28.6.15** `static void GPIO_PortEnableInterrupts ( GPIO_Type * base, uint32_t mask  
) [inline], [static]`

Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | GPIO base pointer.     |
| <i>mask</i> | GPIO pin number macro. |

**28.6.16** `static void GPIO_EnableInterrupts ( GPIO_Type * base, uint32_t mask )`  
**[inline], [static]**

Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | GPIO base pointer.     |
| <i>mask</i> | GPIO pin number macro. |

**28.6.17** `static void GPIO_PortDisableInterrupts ( GPIO_Type * base, uint32_t mask )`  
**[inline], [static]**

Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | GPIO base pointer.     |
| <i>mask</i> | GPIO pin number macro. |

**28.6.18** `static void GPIO_DisableInterrupts ( GPIO_Type * base, uint32_t mask )`  
**[inline], [static]**

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PortDisableInterrupts](#).

**28.6.19** `static uint32_t GPIO_PortGetInterruptFlags ( GPIO_Type * base )`  
**[inline], [static]**

Parameters

---

|             |                    |
|-------------|--------------------|
| <i>base</i> | GPIO base pointer. |
|-------------|--------------------|

Return values

|                |                            |
|----------------|----------------------------|
| <i>current</i> | pin interrupt status flag. |
|----------------|----------------------------|

**28.6.20** `static uint32_t GPIO_GetPinsInterruptFlags ( GPIO_Type * base )`  
**[inline], [static]**

Parameters

|             |                    |
|-------------|--------------------|
| <i>base</i> | GPIO base pointer. |
|-------------|--------------------|

Return values

|                |                            |
|----------------|----------------------------|
| <i>current</i> | pin interrupt status flag. |
|----------------|----------------------------|

**28.6.21** `static void GPIO_PortClearInterruptFlags ( GPIO_Type * base, uint32_t`  
***mask* ) [inline], [static]**

Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | GPIO base pointer.     |
| <i>mask</i> | GPIO pin number macro. |

**28.6.22** `static void GPIO_ClearPinsInterruptFlags ( GPIO_Type * base, uint32_t`  
***mask* ) [inline], [static]**

Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

---

|             |                        |
|-------------|------------------------|
| <i>base</i> | GPIO base pointer.     |
| <i>mask</i> | GPIO pin number macro. |

## Chapter 29

# KPP: KeyPad Port Driver

### 29.1 Overview

The MCUXpresso SDK provides a peripheral driver for the KeyPad Port block of MCUXpresso SDK devices.

## KPP: KeyPad Port Driver

### KPP Initialization Operation

The KPP Initialize is to initialize for common configure: gate the KPP clock, configure columns, and rows features. The KPP Deinitialize is to ungate the clock.

### KPP Basic Operation

The KPP provide the function to enable/disable interrupts. The KPP provide key press scanning function `KPP_keyPressScanning`. This API should be called by the Interrupt handler in application. KPP still provides functions to get and clear status flags.

### 29.2 Typical use case

#### Data Structures

- struct `kpp_config_t`  
*Lists of KPP status. [More...](#)*

#### Enumerations

- enum `kpp_interrupt_enable_t` {  
`kKPP_keyDepressInterrupt` = `KPP_KPSR_KDIE_MASK`,  
`kKPP_keyReleaseInterrupt` = `KPP_KPSR_KRIE_MASK` }  
*List of interrupts supported by the peripheral.*
- enum `kpp_sync_operation_t` {  
`kKPP_ClearKeyDepressSyncChain` = `KPP_KPSR_KDSC_MASK`,  
`kKPP_SetKeyReleasesSyncChain` = `KPP_KPSR_KRSS_MASK` }  
*Lists of KPP synchronize chain operation.*

#### Driver version

- #define `FSL_KPP_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 0)`)  
*KPP driver version 2.0.0.*

## Initialization and De-initialization

- void [KPP\\_Init](#) (KPP\_Type \*base, [kpp\\_config\\_t](#) \*configure)  
*KPP initialize.*
- void [KPP\\_Deinit](#) (KPP\_Type \*base)  
*Deinitializes the KPP module and gates the clock.*

## KPP Basic Operation

- static void [KPP\\_EnableInterrupts](#) (KPP\_Type \*base, uint16\_t mask)  
*Enable the interrupt.*
- static void [KPP\\_DisableInterrupts](#) (KPP\_Type \*base, uint16\_t mask)  
*Disable the interrupt.*
- static uint16\_t [KPP\\_GetStatusFlag](#) (KPP\_Type \*base)  
*Gets the KPP interrupt event status.*
- static void [KPP\\_ClearStatusFlag](#) (KPP\_Type \*base, uint16\_t mask)  
*Clears KPP status flag.*
- static void [KPP\\_SetSynchronizeChain](#) (KPP\_Type \*base, uint16\_t mask)  
*Set KPP synchronization chain.*
- void [KPP\\_keyPressScanning](#) (KPP\_Type \*base, uint8\_t \*data, uint32\_t clockSrc\_Hz)  
*Keypad press scanning.*

## 29.3 Data Structure Documentation

### 29.3.1 struct [kpp\\_config\\_t](#)

#### Data Fields

- uint8\_t [activeRow](#)  
*The row number: bit 7 ~ 0 represents the row 7 ~ 0.*
- uint8\_t [activeColumn](#)  
*The column number: bit 7 ~ 0 represents the column 7 ~ 0.*
- uint16\_t [interrupt](#)  
*KPP interrupt source.*

#### Field Documentation

- (1) [uint8\\_t kpp\\_config\\_t::activeRow](#)
- (2) [uint8\\_t kpp\\_config\\_t::activeColumn](#)
- (3) [uint16\\_t kpp\\_config\\_t::interrupt](#)

A logical OR of "kpp\_interrupt\_enable\_t".

## 29.4 Macro Definition Documentation

### 29.4.1 [#define FSL\\_KPP\\_DRIVER\\_VERSION \(MAKE\\_VERSION\(2, 0, 0\)\)](#)



## 29.5 Enumeration Type Documentation

### 29.5.1 enum kpp\_interrupt\_enable\_t

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

Enumerator

*kKPP\_keyDepressInterrupt* Keypad depress interrupt source.  
*kKPP\_keyReleaseInterrupt* Keypad release interrupt source.

### 29.5.2 enum kpp\_sync\_operation\_t

Enumerator

*kKPP\_ClearKeyDepressSyncChain* Keypad depress interrupt status.  
*kKPP\_SetKeyReleasesSyncChain* Keypad release interrupt status.

## 29.6 Function Documentation

### 29.6.1 void KPP\_Init ( KPP\_Type \* *base*, kpp\_config\_t \* *configure* )

This function ungates the KPP clock and initializes KPP. This function must be called before calling any other KPP driver functions.

Parameters

|                  |                                          |
|------------------|------------------------------------------|
| <i>base</i>      | KPP peripheral base address.             |
| <i>configure</i> | The KPP configuration structure pointer. |

### 29.6.2 void KPP\_Deinit ( KPP\_Type \* *base* )

This function gates the KPP clock. As a result, the KPP module doesn't work after calling this function.

Parameters

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

### 29.6.3 static void KPP\_EnableInterrupts ( KPP\_Type \* *base*, uint16\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                                              |
|-------------|----------------------------------------------------------------------------------------------|
| <i>base</i> | KPP peripheral base address.                                                                 |
| <i>mask</i> | KPP interrupts to enable. This is a logical OR of the enumeration :: kpp_interrupt_enable_t. |

#### 29.6.4 static void KPP\_DisableInterrupts ( KPP\_Type \* *base*, uint16\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                                               |
|-------------|-----------------------------------------------------------------------------------------------|
| <i>base</i> | KPP peripheral base address.                                                                  |
| <i>mask</i> | KPP interrupts to disable. This is a logical OR of the enumeration :: kpp_interrupt_enable_t. |

#### 29.6.5 static uint16\_t KPP\_GetStatusFlag ( KPP\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

The status of the KPP. Application can use the enum type in the "kpp\_interrupt\_enable\_t" to get the right status of the related event.

#### 29.6.6 static void KPP\_ClearStatusFlag ( KPP\_Type \* *base*, uint16\_t *mask* ) [inline], [static]

## Parameters

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

|             |                                                                                            |
|-------------|--------------------------------------------------------------------------------------------|
| <i>mask</i> | KPP mask to be cleared. This is a logical OR of the enumeration :: kpp_interrupt_enable_t. |
|-------------|--------------------------------------------------------------------------------------------|

### 29.6.7 static void KPP\_SetSynchronizeChain ( KPP\_Type \* *base*, uint16\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                          |
|-------------|------------------------------------------------------------------------------------------|
| <i>base</i> | KPP peripheral base address.                                                             |
| <i>mask</i> | KPP mask to be cleared. This is a logical OR of the enumeration :: kpp_sync_operation_t. |

### 29.6.8 void KPP\_keyPressScanning ( KPP\_Type \* *base*, uint8\_t \* *data*, uint32\_t *clockSrc\_Hz* )

This function will scanning all columns and rows. so all scanning data will be stored in the data pointer.

Parameters

|                    |                                                                                                                                                                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | KPP peripheral base address.                                                                                                                                                                                                                                                                                                |
| <i>data</i>        | KPP key press scanning data. The data buffer should be prepared with length at least equal to KPP_KEYPAD_COLUMNNUM_MAX * KPP_KEYPAD_ROWNUM_MAX. the data pointer is recommended to be a array like uint8_t data[KPP_KEYPAD_COLUMNNUM_MAX]. for example the data[2] = 4, that means in column 1 row 2 has a key press event. |
| <i>clockSrc_Hz</i> | Source clock.                                                                                                                                                                                                                                                                                                               |

## Chapter 30

# LPI2C: Low Power Inter-Integrated Circuit Driver

### 30.1 Overview

#### Modules

- [LPI2C CMSIS Driver](#)
- [LPI2C FreeRTOS Driver](#)
- [LPI2C Master DMA Driver](#)
- [LPI2C Master Driver](#)
- [LPI2C Slave Driver](#)

#### Macros

- `#define I2C_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`  
*Retry times for waiting flag.*

#### Enumerations

- `enum {`  
`kStatus_LPI2C_Busy = MAKE_STATUS(kStatusGroup_LPI2C, 0),`  
`kStatus_LPI2C_Idle = MAKE_STATUS(kStatusGroup_LPI2C, 1),`  
`kStatus_LPI2C_Nak = MAKE_STATUS(kStatusGroup_LPI2C, 2),`  
`kStatus_LPI2C_FifoError = MAKE_STATUS(kStatusGroup_LPI2C, 3),`  
`kStatus_LPI2C_BitError = MAKE_STATUS(kStatusGroup_LPI2C, 4),`  
`kStatus_LPI2C_ArbitrationLost = MAKE_STATUS(kStatusGroup_LPI2C, 5),`  
`kStatus_LPI2C_PinLowTimeout,`  
`kStatus_LPI2C_NoTransferInProgress,`  
`kStatus_LPI2C_DmaRequestFail = MAKE_STATUS(kStatusGroup_LPI2C, 8),`  
`kStatus_LPI2C_Timeout = MAKE_STATUS(kStatusGroup_LPI2C, 9) }`  
*LPI2C status return codes.*

#### Driver version

- `#define FSL_LPI2C_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))`  
*LPI2C driver version.*

## 30.2 Macro Definition Documentation

30.2.1 `#define FSL_LPI2C_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))`

30.2.2 `#define I2C_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`

## 30.3 Enumeration Type Documentation

### 30.3.1 anonymous enum

Enumerator

*kStatus\_LPI2C\_Busy* The master is already performing a transfer.

*kStatus\_LPI2C\_Idle* The slave driver is idle.

*kStatus\_LPI2C\_Nak* The slave device sent a NAK in response to a byte.

*kStatus\_LPI2C\_FifoError* FIFO under run or overrun.

*kStatus\_LPI2C\_BitError* Transferred bit was not seen on the bus.

*kStatus\_LPI2C\_ArbitrationLost* Arbitration lost error.

*kStatus\_LPI2C\_PinLowTimeout* SCL or SDA were held low longer than the timeout.

*kStatus\_LPI2C\_NoTransferInProgress* Attempt to abort a transfer when one is not in progress.

*kStatus\_LPI2C\_DmaRequestFail* DMA request failed.

*kStatus\_LPI2C\_Timeout* Timeout polling status flags.

## 30.4 LPI2C Master Driver

### 30.4.1 Overview

#### Data Structures

- struct `lpi2c_master_config_t`  
*Structure with settings to initialize the LPI2C master module. [More...](#)*
- struct `lpi2c_data_match_config_t`  
*LPI2C master data match configuration structure. [More...](#)*
- struct `lpi2c_master_transfer_t`  
*Non-blocking transfer descriptor structure. [More...](#)*
- struct `lpi2c_master_handle_t`  
*Driver handle for master non-blocking APIs. [More...](#)*

#### Typedefs

- typedef `void(* lpi2c_master_transfer_callback_t )(LPI2C_Type *base, lpi2c_master_handle_t *handle, status_t completionStatus, void *userData)`  
*Master completion callback function pointer type.*
- typedef `void(* lpi2c_master_isr_t )(LPI2C_Type *base, void *handle)`  
*Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.*

#### Enumerations

- enum `_lpi2c_master_flags` {  
`kLPI2C_MasterTxReadyFlag = LPI2C_MSR_TDF_MASK,`  
`kLPI2C_MasterRxReadyFlag = LPI2C_MSR_RDF_MASK,`  
`kLPI2C_MasterEndOfPacketFlag = LPI2C_MSR_EPF_MASK,`  
`kLPI2C_MasterStopDetectFlag = LPI2C_MSR_SDF_MASK,`  
`kLPI2C_MasterNackDetectFlag = LPI2C_MSR_NDF_MASK,`  
`kLPI2C_MasterArbitrationLostFlag = LPI2C_MSR_ALF_MASK,`  
`kLPI2C_MasterFifoErrFlag = LPI2C_MSR_FEF_MASK,`  
`kLPI2C_MasterPinLowTimeoutFlag = LPI2C_MSR_PLTF_MASK,`  
`kLPI2C_MasterDataMatchFlag = LPI2C_MSR_DMF_MASK,`  
`kLPI2C_MasterBusyFlag = LPI2C_MSR_MBF_MASK,`  
`kLPI2C_MasterBusBusyFlag = LPI2C_MSR_BBF_MASK,`  
`kLPI2C_MasterClearFlags,`  
`kLPI2C_MasterIrqFlags,`  
`kLPI2C_MasterErrorFlags }`  
*LPI2C master peripheral flags.*
- enum `lpi2c_direction_t` {  
`kLPI2C_Write = 0U,`  
`kLPI2C_Read = 1U }`

- Direction of master and slave transfers.*

    - enum `lpi2c_master_pin_config_t` {  
`kLPI2C_2PinOpenDrain = 0x0U`,  
`kLPI2C_2PinOutputOnly = 0x1U`,  
`kLPI2C_2PinPushPull = 0x2U`,  
`kLPI2C_4PinPushPull = 0x3U`,  
`kLPI2C_2PinOpenDrainWithSeparateSlave`,  
`kLPI2C_2PinOutputOnlyWithSeparateSlave`,  
`kLPI2C_2PinPushPullWithSeparateSlave`,  
`kLPI2C_4PinPushPullWithInvertedOutput = 0x7U` }
  - LPI2C pin configuration.*

    - enum `lpi2c_host_request_source_t` {  
`kLPI2C_HostRequestExternalPin = 0x0U`,  
`kLPI2C_HostRequestInputTrigger = 0x1U` }
  - LPI2C master host request selection.*

    - enum `lpi2c_host_request_polarity_t` {  
`kLPI2C_HostRequestPinActiveLow = 0x0U`,  
`kLPI2C_HostRequestPinActiveHigh = 0x1U` }
  - LPI2C master host request pin polarity configuration.*

    - enum `lpi2c_data_match_config_mode_t` {  
`kLPI2C_MatchDisabled = 0x0U`,  
`kLPI2C_1stWordEqualsM0OrM1 = 0x2U`,  
`kLPI2C_AnyWordEqualsM0OrM1 = 0x3U`,  
`kLPI2C_1stWordEqualsM0And2ndWordEqualsM1`,  
`kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1`,  
`kLPI2C_1stWordAndM1EqualsM0AndM1`,  
`kLPI2C_AnyWordAndM1EqualsM0AndM1` }
  - LPI2C master data match configuration modes.*

    - enum `_lpi2c_master_transfer_flags` {  
`kLPI2C_TransferDefaultFlag = 0x00U`,  
`kLPI2C_TransferNoStartFlag = 0x01U`,  
`kLPI2C_TransferRepeatedStartFlag = 0x02U`,  
`kLPI2C_TransferNoStopFlag = 0x04U` }
- Transfer option flags.*

## Initialization and deinitialization

- void `LPI2C_MasterGetDefaultConfig` (`lpi2c_master_config_t *masterConfig`)  
*Provides a default configuration for the LPI2C master peripheral.*
- void `LPI2C_MasterInit` (`LPI2C_Type *base`, `const lpi2c_master_config_t *masterConfig`, `uint32_t sourceClock_Hz`)  
*Initializes the LPI2C master peripheral.*
- void `LPI2C_MasterDeinit` (`LPI2C_Type *base`)  
*Deinitializes the LPI2C master peripheral.*
- void `LPI2C_MasterConfigureDataMatch` (`LPI2C_Type *base`, `const lpi2c_data_match_config_t *matchConfig`)

*Configures LPI2C master data match feature.*

- `status_t LPI2C_MasterCheckAndClearError` (LPI2C\_Type \*base, uint32\_t status)
- `status_t LPI2C_CheckForBusyBus` (LPI2C\_Type \*base)
- `static void LPI2C_MasterReset` (LPI2C\_Type \*base)

*Performs a software reset.*

- `static void LPI2C_MasterEnable` (LPI2C\_Type \*base, bool enable)  
*Enables or disables the LPI2C module as master.*

## Status

- `static uint32_t LPI2C_MasterGetStatusFlags` (LPI2C\_Type \*base)  
*Gets the LPI2C master status flags.*
- `static void LPI2C_MasterClearStatusFlags` (LPI2C\_Type \*base, uint32\_t statusMask)  
*Clears the LPI2C master status flag state.*

## Interrupts

- `static void LPI2C_MasterEnableInterrupts` (LPI2C\_Type \*base, uint32\_t interruptMask)  
*Enables the LPI2C master interrupt requests.*
- `static void LPI2C_MasterDisableInterrupts` (LPI2C\_Type \*base, uint32\_t interruptMask)  
*Disables the LPI2C master interrupt requests.*
- `static uint32_t LPI2C_MasterGetEnabledInterrupts` (LPI2C\_Type \*base)  
*Returns the set of currently enabled LPI2C master interrupt requests.*

## DMA control

- `static void LPI2C_MasterEnableDMA` (LPI2C\_Type \*base, bool enableTx, bool enableRx)  
*Enables or disables LPI2C master DMA requests.*
- `static uint32_t LPI2C_MasterGetTxFifoAddress` (LPI2C\_Type \*base)  
*Gets LPI2C master transmit data register address for DMA transfer.*
- `static uint32_t LPI2C_MasterGetRxFifoAddress` (LPI2C\_Type \*base)  
*Gets LPI2C master receive data register address for DMA transfer.*

## FIFO control

- `static void LPI2C_MasterSetWatermarks` (LPI2C\_Type \*base, size\_t txWords, size\_t rxWords)  
*Sets the watermarks for LPI2C master FIFOs.*
- `static void LPI2C_MasterGetFifoCounts` (LPI2C\_Type \*base, size\_t \*rxCount, size\_t \*txCount)  
*Gets the current number of words in the LPI2C master FIFOs.*

## Bus operations

- `void LPI2C_MasterSetBaudRate` (LPI2C\_Type \*base, uint32\_t sourceClock\_Hz, uint32\_t baudRate\_Hz)



- *Sets the I2C bus frequency for master transactions.*
- static bool [LPI2C\\_MasterGetBusIdleState](#) (LPI2C\_Type \*base)  
*Returns whether the bus is idle.*
- [status\\_t LPI2C\\_MasterStart](#) (LPI2C\_Type \*base, uint8\_t address, [lpi2c\\_direction\\_t](#) dir)  
*Sends a START signal and slave address on the I2C bus.*
- static [status\\_t LPI2C\\_MasterRepeatedStart](#) (LPI2C\_Type \*base, uint8\_t address, [lpi2c\\_direction\\_t](#) dir)  
*Sends a repeated START signal and slave address on the I2C bus.*
- [status\\_t LPI2C\\_MasterSend](#) (LPI2C\_Type \*base, void \*txBuff, size\_t txSize)  
*Performs a polling send transfer on the I2C bus.*
- [status\\_t LPI2C\\_MasterReceive](#) (LPI2C\_Type \*base, void \*rxBuff, size\_t rxSize)  
*Performs a polling receive transfer on the I2C bus.*
- [status\\_t LPI2C\\_MasterStop](#) (LPI2C\_Type \*base)  
*Sends a STOP signal on the I2C bus.*
- [status\\_t LPI2C\\_MasterTransferBlocking](#) (LPI2C\_Type \*base, [lpi2c\\_master\\_transfer\\_t](#) \*transfer)  
*Performs a master polling transfer on the I2C bus.*

## Non-blocking

- void [LPI2C\\_MasterTransferCreateHandle](#) (LPI2C\_Type \*base, [lpi2c\\_master\\_handle\\_t](#) \*handle, [lpi2c\\_master\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Creates a new handle for the LPI2C master non-blocking APIs.*
- [status\\_t LPI2C\\_MasterTransferNonBlocking](#) (LPI2C\_Type \*base, [lpi2c\\_master\\_handle\\_t](#) \*handle, [lpi2c\\_master\\_transfer\\_t](#) \*transfer)  
*Performs a non-blocking transaction on the I2C bus.*
- [status\\_t LPI2C\\_MasterTransferGetCount](#) (LPI2C\_Type \*base, [lpi2c\\_master\\_handle\\_t](#) \*handle, size\_t \*count)  
*Returns number of bytes transferred so far.*
- void [LPI2C\\_MasterTransferAbort](#) (LPI2C\_Type \*base, [lpi2c\\_master\\_handle\\_t](#) \*handle)  
*Terminates a non-blocking LPI2C master transmission early.*

## IRQ handler

- void [LPI2C\\_MasterTransferHandleIRQ](#) (LPI2C\_Type \*base, void \*lpi2cMasterHandle)  
*Reusable routine to handle master interrupts.*

## 30.4.2 Data Structure Documentation

### 30.4.2.1 struct [lpi2c\\_master\\_config\\_t](#)

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the [LPI2C\\_MasterGetDefaultConfig\(\)](#) function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

## Data Fields

- bool `enableMaster`  
*Whether to enable master mode.*
- bool `enableDoze`  
*Whether master is enabled in doze mode.*
- bool `debugEnable`  
*Enable transfers to continue when halted in debug mode.*
- bool `ignoreAck`  
*Whether to ignore ACK/NACK.*
- `lpi2c_master_pin_config_t` `pinConfig`  
*The pin configuration option.*
- `uint32_t` `baudRate_Hz`  
*Desired baud rate in Hertz.*
- `uint32_t` `busIdleTimeout_ns`  
*Bus idle timeout in nanoseconds.*
- `uint32_t` `pinLowTimeout_ns`  
*Pin low timeout in nanoseconds.*
- `uint8_t` `sdaGlitchFilterWidth_ns`  
*Width in nanoseconds of glitch filter on SDA pin.*
- `uint8_t` `sclGlitchFilterWidth_ns`  
*Width in nanoseconds of glitch filter on SCL pin.*
- struct {
  - bool `enable`  
*Enable host request.*
  - `lpi2c_host_request_source_t` `source`  
*Host request source.*
  - `lpi2c_host_request_polarity_t` `polarity`  
*Host request pin polarity.*
- } `hostRequest`  
  
*Host request options.*

## Field Documentation

- (1) `bool lpi2c_master_config_t::enableMaster`
- (2) `bool lpi2c_master_config_t::enableDoze`
- (3) `bool lpi2c_master_config_t::debugEnable`
- (4) `bool lpi2c_master_config_t::ignoreAck`
- (5) `lpi2c_master_pin_config_t lpi2c_master_config_t::pinConfig`
- (6) `uint32_t lpi2c_master_config_t::baudRate_Hz`
- (7) `uint32_t lpi2c_master_config_t::busIdleTimeout_ns`

Set to 0 to disable.

**(8) uint32\_t lpi2c\_master\_config\_t::pinLowTimeout\_ns**

Set to 0 to disable.

**(9) uint8\_t lpi2c\_master\_config\_t::sdaGlitchFilterWidth\_ns**

Set to 0 to disable.

**(10) uint8\_t lpi2c\_master\_config\_t::sclGlitchFilterWidth\_ns**

Set to 0 to disable.

**(11) bool lpi2c\_master\_config\_t::enable****(12) lpi2c\_host\_request\_source\_t lpi2c\_master\_config\_t::source****(13) lpi2c\_host\_request\_polarity\_t lpi2c\_master\_config\_t::polarity****(14) struct { ... } lpi2c\_master\_config\_t::hostRequest****30.4.2.2 struct lpi2c\_data\_match\_config\_t****Data Fields**

- [lpi2c\\_data\\_match\\_config\\_mode\\_t matchMode](#)  
*Data match configuration setting.*
- [bool rxDataMatchOnly](#)  
*When set to true, received data is ignored until a successful match.*
- [uint32\\_t match0](#)  
*Match value 0.*
- [uint32\\_t match1](#)  
*Match value 1.*

**Field Documentation****(1) lpi2c\_data\_match\_config\_mode\_t lpi2c\_data\_match\_config\_t::matchMode****(2) bool lpi2c\_data\_match\_config\_t::rxDataMatchOnly****(3) uint32\_t lpi2c\_data\_match\_config\_t::match0****(4) uint32\_t lpi2c\_data\_match\_config\_t::match1****30.4.2.3 struct lpi2c\_master\_transfer**

This structure is used to pass transaction parameters to the [LPI2C\\_MasterTransferNonBlocking\(\)](#) API.

**Data Fields**

- [uint32\\_t flags](#)

- *Bit mask of options for the transfer.*
- `uint16_t slaveAddress`  
*The 7-bit slave address.*
- `lpi2c_direction_t direction`  
*Either `kLPI2C_Read` or `kLPI2C_Write`.*
- `uint32_t subaddress`  
*Sub address.*
- `size_t subaddressSize`  
*Length of sub address to send in bytes.*
- `void * data`  
*Pointer to data to transfer.*
- `size_t dataSize`  
*Number of bytes to transfer.*

### Field Documentation

#### (1) `uint32_t lpi2c_master_transfer_t::flags`

See enumeration `_lpi2c_master_transfer_flags` for available options. Set to 0 or `kLPI2C_TransferDefaultFlag` for normal transfers.

#### (2) `uint16_t lpi2c_master_transfer_t::slaveAddress`

#### (3) `lpi2c_direction_t lpi2c_master_transfer_t::direction`

#### (4) `uint32_t lpi2c_master_transfer_t::subaddress`

Transferred MSB first.

#### (5) `size_t lpi2c_master_transfer_t::subaddressSize`

Maximum size is 4 bytes.

#### (6) `void* lpi2c_master_transfer_t::data`

#### (7) `size_t lpi2c_master_transfer_t::dataSize`

### 30.4.2.4 `struct _lpi2c_master_handle`

Note

The contents of this structure are private and subject to change.

### Data Fields

- `uint8_t state`  
*Transfer state machine current state.*
- `uint16_t remainingBytes`  
*Remaining byte count in current state.*
- `uint8_t * buf`

- *Buffer pointer for current state.*  
uint16\_t `commandBuffer` [6]
- *LPI2C command sequence.*  
lpi2c\_master\_transfer\_t `transfer`
- *Copy of the current transfer info.*  
lpi2c\_master\_transfer\_callback\_t `completionCallback`
- *Callback function pointer.*  
void \* `userData`
- *Application data passed to callback.*

### Field Documentation

- (1) `uint8_t lpi2c_master_handle_t::state`
- (2) `uint16_t lpi2c_master_handle_t::remainingBytes`
- (3) `uint8_t* lpi2c_master_handle_t::buf`
- (4) `uint16_t lpi2c_master_handle_t::commandBuffer[6]`

When all 6 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]

- (5) `lpi2c_master_transfer_t lpi2c_master_handle_t::transfer`
- (6) `lpi2c_master_transfer_callback_t lpi2c_master_handle_t::completionCallback`
- (7) `void* lpi2c_master_handle_t::userData`

### 30.4.3 Typedef Documentation

#### 30.4.3.1 `typedef void(* lpi2c_master_transfer_callback_t)(LPI2C_Type *base, lpi2c_master_handle_t *handle, status_t completionStatus, void *userData)`

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to `LPI2C_MasterTransferCreateHandle()`.

Parameters

|                          |                                                                                             |
|--------------------------|---------------------------------------------------------------------------------------------|
| <i>base</i>              | The LPI2C peripheral base address.                                                          |
| <i>completion-Status</i> | Either <code>kStatus_Success</code> or an error code describing how the transfer completed. |

|                 |                                                            |
|-----------------|------------------------------------------------------------|
| <i>userData</i> | Arbitrary pointer-sized value passed from the application. |
|-----------------|------------------------------------------------------------|

### 30.4.4 Enumeration Type Documentation

#### 30.4.4.1 enum `_lpi2c_master_flags`

The following status register flags can be cleared:

- `kLPI2C_MasterEndOfPacketFlag`
- `kLPI2C_MasterStopDetectFlag`
- `kLPI2C_MasterNackDetectFlag`
- `kLPI2C_MasterArbitrationLostFlag`
- `kLPI2C_MasterFifoErrFlag`
- `kLPI2C_MasterPinLowTimeoutFlag`
- `kLPI2C_MasterDataMatchFlag`

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be enabled as interrupts.

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

- `kLPI2C_MasterTxReadyFlag`*** Transmit data flag.
- `kLPI2C_MasterRxReadyFlag`*** Receive data flag.
- `kLPI2C_MasterEndOfPacketFlag`*** End Packet flag.
- `kLPI2C_MasterStopDetectFlag`*** Stop detect flag.
- `kLPI2C_MasterNackDetectFlag`*** NACK detect flag.
- `kLPI2C_MasterArbitrationLostFlag`*** Arbitration lost flag.
- `kLPI2C_MasterFifoErrFlag`*** FIFO error flag.
- `kLPI2C_MasterPinLowTimeoutFlag`*** Pin low timeout flag.
- `kLPI2C_MasterDataMatchFlag`*** Data match flag.
- `kLPI2C_MasterBusyFlag`*** Master busy flag.
- `kLPI2C_MasterBusBusyFlag`*** Bus busy flag.
- `kLPI2C_MasterClearFlags`*** All flags which are cleared by the driver upon starting a transfer.
- `kLPI2C_MasterIrqFlags`*** IRQ sources enabled by the non-blocking transactional API.
- `kLPI2C_MasterErrorFlags`*** Errors to check for.

#### 30.4.4.2 enum `lpi2c_direction_t`

Enumerator

- `kLPI2C_Write`*** Master transmit.
- `kLPI2C_Read`*** Master receive.

### 30.4.4.3 enum `lpi2c_master_pin_config_t`

Enumerator

- kLPI2C\_2PinOpenDrain*** LPI2C Configured for 2-pin open drain mode.
- kLPI2C\_2PinOutputOnly*** LPI2C Configured for 2-pin output only mode (ultra-fast mode)
- kLPI2C\_2PinPushPull*** LPI2C Configured for 2-pin push-pull mode.
- kLPI2C\_4PinPushPull*** LPI2C Configured for 4-pin push-pull mode.
- kLPI2C\_2PinOpenDrainWithSeparateSlave*** LPI2C Configured for 2-pin open drain mode with separate LPI2C slave.
- kLPI2C\_2PinOutputOnlyWithSeparateSlave*** LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave.
- kLPI2C\_2PinPushPullWithSeparateSlave*** LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave.
- kLPI2C\_4PinPushPullWithInvertedOutput*** LPI2C Configured for 4-pin push-pull mode(inverted outputs)

### 30.4.4.4 enum `lpi2c_host_request_source_t`

Enumerator

- kLPI2C\_HostRequestExternalPin*** Select the LPI2C\_HREQ pin as the host request input.
- kLPI2C\_HostRequestInputTrigger*** Select the input trigger as the host request input.

### 30.4.4.5 enum `lpi2c_host_request_polarity_t`

Enumerator

- kLPI2C\_HostRequestPinActiveLow*** Configure the LPI2C\_HREQ pin active low.
- kLPI2C\_HostRequestPinActiveHigh*** Configure the LPI2C\_HREQ pin active high.

### 30.4.4.6 enum `lpi2c_data_match_config_mode_t`

Enumerator

- kLPI2C\_MatchDisabled*** LPI2C Match Disabled.
- kLPI2C\_1stWordEqualsM0OrM1*** LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1.
- kLPI2C\_AnyWordEqualsM0OrM1*** LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1.
- kLPI2C\_1stWordEqualsM0And2ndWordEqualsM1*** LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1.
- kLPI2C\_AnyWordEqualsM0AndNextWordEqualsM1*** LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1.

***kLPI2C\_1stWordAndM1EqualsM0AndM1*** LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1.

***kLPI2C\_AnyWordAndM1EqualsM0AndM1*** LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1.

#### 30.4.4.7 enum \_lpi2c\_master\_transfer\_flags

Note

These enumerations are intended to be OR'd together to form a bit mask of options for the `_lpi2c_master_transfer::flags` field.

Enumerator

***kLPI2C\_TransferDefaultFlag*** Transfer starts with a start signal, stops with a stop signal.

***kLPI2C\_TransferNoStartFlag*** Don't send a start condition, address, and sub address.

***kLPI2C\_TransferRepeatedStartFlag*** Send a repeated start condition.

***kLPI2C\_TransferNoStopFlag*** Don't send a stop condition.

### 30.4.5 Function Documentation

#### 30.4.5.1 void LPI2C\_MasterGetDefaultConfig ( lpi2c\_master\_config\_t \* masterConfig )

This function provides the following default configuration for the LPI2C master peripheral:

```
* masterConfig->enableMaster = true;
* masterConfig->debugEnable = false;
* masterConfig->ignoreAck = false;
* masterConfig->pinConfig = kLPI2C_2PinOpenDrain;
* masterConfig->baudRate_Hz = 100000U;
* masterConfig->busIdleTimeout_ns = 0;
* masterConfig->pinLowTimeout_ns = 0;
* masterConfig->sdaGlitchFilterWidth_ns = 0;
* masterConfig->sclGlitchFilterWidth_ns = 0;
* masterConfig->hostRequest.enable = false;
* masterConfig->hostRequest.source = kLPI2C_HostRequestExternalPin;
* masterConfig->hostRequest.polarity = kLPI2C_HostRequestPinActiveHigh;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with `LPI2C_MasterInit()`.

Parameters

---



|     |                     |                                                                                                            |
|-----|---------------------|------------------------------------------------------------------------------------------------------------|
| out | <i>masterConfig</i> | User provided configuration structure for default values. Refer to <a href="#">lpi2c_master_config_t</a> . |
|-----|---------------------|------------------------------------------------------------------------------------------------------------|

### 30.4.5.2 void LPI2C\_MasterInit ( LPI2C\_Type \* *base*, const lpi2c\_master\_config\_t \* *masterConfig*, uint32\_t *sourceClock\_Hz* )

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

|                       |                                                                                                                                            |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>           | The LPI2C peripheral base address.                                                                                                         |
| <i>masterConfig</i>   | User provided peripheral configuration. Use <a href="#">LPI2C_MasterGetDefaultConfig()</a> to get a set of defaults that you can override. |
| <i>sourceClock_Hz</i> | Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.            |

### 30.4.5.3 void LPI2C\_MasterDeinit ( LPI2C\_Type \* *base* )

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

### 30.4.5.4 void LPI2C\_MasterConfigureDataMatch ( LPI2C\_Type \* *base*, const lpi2c\_data\_match\_config\_t \* *matchConfig* )

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>base</i>        | The LPI2C peripheral base address.   |
| <i>matchConfig</i> | Settings for the data match feature. |

### 30.4.5.5 static void LPI2C\_MasterReset ( LPI2C\_Type \* *base* ) [inline], [static]

Restores the LPI2C master peripheral to reset conditions.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

**30.4.5.6** `static void LPI2C_MasterEnable ( LPI2C_Type * base, bool enable ) [inline], [static]`

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                                     |
| <i>enable</i> | Pass true to enable or false to disable the specified LPI2C as master. |

**30.4.5.7** `static uint32_t LPI2C_MasterGetStatusFlags ( LPI2C_Type * base ) [inline], [static]`

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

See Also

[\\_lpi2c\\_master\\_flags](#)

**30.4.5.8** `static void LPI2C_MasterClearStatusFlags ( LPI2C_Type * base, uint32_t statusMask ) [inline], [static]`

The following status register flags can be cleared:

- [kLPI2C\\_MasterEndOfPacketFlag](#)
- [kLPI2C\\_MasterStopDetectFlag](#)
- [kLPI2C\\_MasterNackDetectFlag](#)
- [kLPI2C\\_MasterArbitrationLostFlag](#)

- [kLPI2C\\_MasterFifoErrFlag](#)
- [kLPI2C\\_MasterPinLowTimeoutFlag](#)
- [kLPI2C\\_MasterDataMatchFlag](#)

Attempts to clear other flags has no effect.

Parameters

|                   |                                                                                                                                                                                                                                    |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | The LPI2C peripheral base address.                                                                                                                                                                                                 |
| <i>statusMask</i> | A bitmask of status flags that are to be cleared. The mask is composed of <code>_lpi2c_master_flags</code> enumerators OR'd together. You may pass the result of a previous call to <a href="#">LPI2C_MasterGetStatusFlags()</a> . |

See Also

[\\_lpi2c\\_master\\_flags](#).

#### 30.4.5.9 `static void LPI2C_MasterEnableInterrupts ( LPI2C_Type * base, uint32_t interruptMask ) [inline], [static]`

All flags except [kLPI2C\\_MasterBusyFlag](#) and [kLPI2C\\_MasterBusBusyFlag](#) can be enabled as interrupts.

Parameters

|                      |                                                                                                                                                       |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The LPI2C peripheral base address.                                                                                                                    |
| <i>interruptMask</i> | Bit mask of interrupts to enable. See <a href="#">_lpi2c_master_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 30.4.5.10 `static void LPI2C_MasterDisableInterrupts ( LPI2C_Type * base, uint32_t interruptMask ) [inline], [static]`

All flags except [kLPI2C\\_MasterBusyFlag](#) and [kLPI2C\\_MasterBusBusyFlag](#) can be enabled as interrupts.

Parameters

|                      |                                                                                                                                                        |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The LPI2C peripheral base address.                                                                                                                     |
| <i>interruptMask</i> | Bit mask of interrupts to disable. See <a href="#">_lpi2c_master_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 30.4.5.11 `static uint32_t LPI2C_MasterGetEnabledInterrupts ( LPI2C_Type * base ) [inline], [static]`

## Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

## Returns

A bitmask composed of `_lpi2c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

**30.4.5.12** `static void LPI2C_MasterEnableDMA ( LPI2C_Type * base, bool enableTx, bool enableRx ) [inline], [static]`

## Parameters

|                 |                                                                                |
|-----------------|--------------------------------------------------------------------------------|
| <i>base</i>     | The LPI2C peripheral base address.                                             |
| <i>enableTx</i> | Enable flag for transmit DMA request. Pass true for enable, false for disable. |
| <i>enableRx</i> | Enable flag for receive DMA request. Pass true for enable, false for disable.  |

**30.4.5.13** `static uint32_t LPI2C_MasterGetTxFifoAddress ( LPI2C_Type * base ) [inline], [static]`

## Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

## Returns

The LPI2C Master Transmit Data Register address.

**30.4.5.14** `static uint32_t LPI2C_MasterGetRxFifoAddress ( LPI2C_Type * base ) [inline], [static]`

## Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

The LPI2C Master Receive Data Register address.

**30.4.5.15** `static void LPI2C_MasterSetWatermarks ( LPI2C_Type * base, size_t txWords, size_t rxWords ) [inline], [static]`

Parameters

|                |                                                                                                                                                                                                                                                          |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The LPI2C peripheral base address.                                                                                                                                                                                                                       |
| <i>txWords</i> | Transmit FIFO watermark value in words. The <code>kLPI2C_MasterTxReadyFlag</code> flag is set whenever the number of words in the transmit FIFO is equal or less than <i>txWords</i> . Writing a value equal or greater than the FIFO size is truncated. |
| <i>rxWords</i> | Receive FIFO watermark value in words. The <code>kLPI2C_MasterRxReadyFlag</code> flag is set whenever the number of words in the receive FIFO is greater than <i>rxWords</i> . Writing a value equal or greater than the FIFO size is truncated.         |

**30.4.5.16** `static void LPI2C_MasterGetFifoCounts ( LPI2C_Type * base, size_t * rxCount, size_t * txCount ) [inline], [static]`

Parameters

|     |                |                                                                                                                              |
|-----|----------------|------------------------------------------------------------------------------------------------------------------------------|
|     | <i>base</i>    | The LPI2C peripheral base address.                                                                                           |
| out | <i>txCount</i> | Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required. |
| out | <i>rxCount</i> | Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.  |

**30.4.5.17** `void LPI2C_MasterSetBaudRate ( LPI2C_Type * base, uint32_t sourceClock_Hz, uint32_t baudRate_Hz )`

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

## Note

Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

## Parameters

|                       |                                            |
|-----------------------|--------------------------------------------|
| <i>base</i>           | The LPI2C peripheral base address.         |
| <i>sourceClock_Hz</i> | LPI2C functional clock frequency in Hertz. |
| <i>baudRate_Hz</i>    | Requested bus frequency in Hertz.          |

### 30.4.5.18 static bool LPI2C\_MasterGetBusIdleState ( LPI2C\_Type \* *base* ) [inline], [static]

Requires the master mode to be enabled.

## Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

## Return values

|              |              |
|--------------|--------------|
| <i>true</i>  | Bus is busy. |
| <i>false</i> | Bus is idle. |

### 30.4.5.19 status\_t LPI2C\_MasterStart ( LPI2C\_Type \* *base*, uint8\_t *address*, lpi2c\_direction\_t *dir* )

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

## Parameters

|                |                                                                                                                                                                                     |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The LPI2C peripheral base address.                                                                                                                                                  |
| <i>address</i> | 7-bit slave device address, in bits [6:0].                                                                                                                                          |
| <i>dir</i>     | Master transfer direction, either <a href="#">kLPI2C_Read</a> or <a href="#">kLPI2C_Write</a> . This parameter is used to set the R/w bit (bit 0) in the transmitted slave address. |

Return values

|                           |                                                                           |
|---------------------------|---------------------------------------------------------------------------|
| <i>kStatus_Success</i>    | START signal and address were successfully enqueued in the transmit FIFO. |
| <i>kStatus_LPI2C_Busy</i> | Another master is currently utilizing the bus.                            |

#### 30.4.5.20 static status\_t LPI2C\_MasterRepeatedStart ( LPI2C\_Type \* base, uint8\_t address, lpi2c\_direction\_t dir ) [inline], [static]

This function is used to send a Repeated START signal when a transfer is already in progress. Like [LPI2C\\_MasterStart\(\)](#), it also sends the specified 7-bit address.

Note

This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

|                |                                                                                                                                                                                     |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The LPI2C peripheral base address.                                                                                                                                                  |
| <i>address</i> | 7-bit slave device address, in bits [6:0].                                                                                                                                          |
| <i>dir</i>     | Master transfer direction, either <a href="#">kLPI2C_Read</a> or <a href="#">kLPI2C_Write</a> . This parameter is used to set the R/w bit (bit 0) in the transmitted slave address. |

Return values

|                           |                                                                                    |
|---------------------------|------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>    | Repeated START signal and address were successfully enqueued in the transmit FIFO. |
| <i>kStatus_LPI2C_Busy</i> | Another master is currently utilizing the bus.                                     |

#### 30.4.5.21 status\_t LPI2C\_MasterSend ( LPI2C\_Type \* base, void \* txBuff, size\_t txSize )

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns [kStatus\\_LPI2C\\_Nak](#).

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                 |
| <i>txBuff</i> | The pointer to the data to be transferred.         |
| <i>txSize</i> | The length in bytes of the data to be transferred. |



Return values

|                                      |                                                    |
|--------------------------------------|----------------------------------------------------|
| <i>kStatus_Success</i>               | Data was sent successfully.                        |
| <i>kStatus_LPI2C_Busy</i>            | Another master is currently utilizing the bus.     |
| <i>kStatus_LPI2C_Nak</i>             | The slave device sent a NAK in response to a byte. |
| <i>kStatus_LPI2C_FifoError</i>       | FIFO under run or over run.                        |
| <i>kStatus_LPI2C_ArbitrationLost</i> | Arbitration lost error.                            |
| <i>kStatus_LPI2C_PinLowTimeout</i>   | SCL or SDA were held low longer than the timeout.  |

#### 30.4.5.22 `status_t LPI2C_MasterReceive ( LPI2C_Type * base, void * rxBuff, size_t rxSize )`

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                 |
| <i>rxBuff</i> | The pointer to the data to be transferred.         |
| <i>rxSize</i> | The length in bytes of the data to be transferred. |

Return values

|                                      |                                                    |
|--------------------------------------|----------------------------------------------------|
| <i>kStatus_Success</i>               | Data was received successfully.                    |
| <i>kStatus_LPI2C_Busy</i>            | Another master is currently utilizing the bus.     |
| <i>kStatus_LPI2C_Nak</i>             | The slave device sent a NAK in response to a byte. |
| <i>kStatus_LPI2C_FifoError</i>       | FIFO under run or overrun.                         |
| <i>kStatus_LPI2C_ArbitrationLost</i> | Arbitration lost error.                            |
| <i>kStatus_LPI2C_PinLowTimeout</i>   | SCL or SDA were held low longer than the timeout.  |

#### 30.4.5.23 `status_t LPI2C_MasterStop ( LPI2C_Type * base )`

This function does not return until the STOP signal is seen on the bus, or an error occurs.

## Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

## Return values

|                                      |                                                                                  |
|--------------------------------------|----------------------------------------------------------------------------------|
| <i>kStatus_Success</i>               | The STOP signal was successfully sent on the bus and the transaction terminated. |
| <i>kStatus_LPI2C_Busy</i>            | Another master is currently utilizing the bus.                                   |
| <i>kStatus_LPI2C_Nak</i>             | The slave device sent a NAK in response to a byte.                               |
| <i>kStatus_LPI2C_FifoError</i>       | FIFO under run or overrun.                                                       |
| <i>kStatus_LPI2C_ArbitrationLost</i> | Arbitration lost error.                                                          |
| <i>kStatus_LPI2C_PinLowTimeout</i>   | SCL or SDA were held low longer than the timeout.                                |

#### 30.4.5.24 status\_t LPI2C\_MasterTransferBlocking ( LPI2C\_Type \* *base*, lpi2c\_master\_transfer\_t \* *transfer* )

## Note

The API does not return until the transfer succeeds or fails due to error happens during transfer.

## Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>base</i>     | The LPI2C peripheral base address. |
| <i>transfer</i> | Pointer to the transfer structure. |

## Return values

|                                      |                                                    |
|--------------------------------------|----------------------------------------------------|
| <i>kStatus_Success</i>               | Data was received successfully.                    |
| <i>kStatus_LPI2C_Busy</i>            | Another master is currently utilizing the bus.     |
| <i>kStatus_LPI2C_Nak</i>             | The slave device sent a NAK in response to a byte. |
| <i>kStatus_LPI2C_FifoError</i>       | FIFO under run or overrun.                         |
| <i>kStatus_LPI2C_ArbitrationLost</i> | Arbitration lost error.                            |
| <i>kStatus_LPI2C_PinLowTimeout</i>   | SCL or SDA were held low longer than the timeout.  |

**30.4.5.25** void LPI2C\_MasterTransferCreateHandle ( LPI2C\_Type \* *base*,  
lpi2c\_master\_handle\_t \* *handle*, lpi2c\_master\_transfer\_callback\_t *callback*,  
void \* *userData* )

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [LPI2C\\_MasterTransferAbort\(\)](#) API shall be called.

Note

The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

|     |                 |                                                              |
|-----|-----------------|--------------------------------------------------------------|
|     | <i>base</i>     | The LPI2C peripheral base address.                           |
| out | <i>handle</i>   | Pointer to the LPI2C master driver handle.                   |
|     | <i>callback</i> | User provided pointer to the asynchronous callback function. |
|     | <i>userData</i> | User provided pointer to the application callback data.      |

**30.4.5.26** status\_t LPI2C\_MasterTransferNonBlocking ( LPI2C\_Type \* *base*,  
lpi2c\_master\_handle\_t \* *handle*, lpi2c\_master\_transfer\_t \* *transfer* )

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>base</i>     | The LPI2C peripheral base address.         |
| <i>handle</i>   | Pointer to the LPI2C master driver handle. |
| <i>transfer</i> | The pointer to the transfer descriptor.    |

Return values

|                           |                                                                                                             |
|---------------------------|-------------------------------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>    | The transaction was started successfully.                                                                   |
| <i>kStatus_LPI2C_Busy</i> | Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress. |

**30.4.5.27** status\_t LPI2C\_MasterTransferGetCount ( LPI2C\_Type \* *base*,  
lpi2c\_master\_handle\_t \* *handle*, size\_t \* *count* )

## Parameters

|     |               |                                                                     |
|-----|---------------|---------------------------------------------------------------------|
|     | <i>base</i>   | The LPI2C peripheral base address.                                  |
|     | <i>handle</i> | Pointer to the LPI2C master driver handle.                          |
| out | <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

## Return values

|                                     |                                                                |
|-------------------------------------|----------------------------------------------------------------|
| <i>kStatus_Success</i>              |                                                                |
| <i>kStatus_NoTransferInProgress</i> | There is not a non-blocking transaction currently in progress. |

### 30.4.5.28 void LPI2C\_MasterTransferAbort ( LPI2C\_Type \* *base*, lpi2c\_master\_handle\_t \* *handle* )

## Note

It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

## Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.         |
| <i>handle</i> | Pointer to the LPI2C master driver handle. |

## Return values

|                           |                                                                |
|---------------------------|----------------------------------------------------------------|
| <i>kStatus_Success</i>    | A transaction was successfully aborted.                        |
| <i>kStatus_LPI2C_Idle</i> | There is not a non-blocking transaction currently in progress. |

### 30.4.5.29 void LPI2C\_MasterTransferHandleIRQ ( LPI2C\_Type \* *base*, void \* *lpi2cMasterHandle* )

## Note

This function does not need to be called unless you are reimplementing the nonblocking API's interrupt handler routines to add special functionality.

## Parameters

|                           |                                            |
|---------------------------|--------------------------------------------|
| <i>base</i>               | The LPI2C peripheral base address.         |
| <i>lpi2cMaster-Handle</i> | Pointer to the LPI2C master driver handle. |

## 30.5 LPI2C Slave Driver

### 30.5.1 Overview

#### Data Structures

- struct `lpi2c_slave_config_t`  
*Structure with settings to initialize the LPI2C slave module. [More...](#)*
- struct `lpi2c_slave_transfer_t`  
*LPI2C slave transfer structure. [More...](#)*
- struct `lpi2c_slave_handle_t`  
*LPI2C slave handle structure. [More...](#)*

#### Typedefs

- typedef `void(* lpi2c_slave_transfer_callback_t )(LPI2C_Type *base, lpi2c_slave_transfer_t *transfer, void *userData)`  
*Slave event callback function pointer type.*

#### Enumerations

- enum `_lpi2c_slave_flags` {  
`kLPI2C_SlaveTxReadyFlag = LPI2C_SSR_TDF_MASK,`  
`kLPI2C_SlaveRxReadyFlag = LPI2C_SSR_RDF_MASK,`  
`kLPI2C_SlaveAddressValidFlag = LPI2C_SSR_AVF_MASK,`  
`kLPI2C_SlaveTransmitAckFlag = LPI2C_SSR_TAF_MASK,`  
`kLPI2C_SlaveRepeatedStartDetectFlag = LPI2C_SSR_RSF_MASK,`  
`kLPI2C_SlaveStopDetectFlag = LPI2C_SSR_SDF_MASK,`  
`kLPI2C_SlaveBitErrFlag = LPI2C_SSR_BEF_MASK,`  
`kLPI2C_SlaveFifoErrFlag = LPI2C_SSR_FEF_MASK,`  
`kLPI2C_SlaveAddressMatch0Flag = LPI2C_SSR_AM0F_MASK,`  
`kLPI2C_SlaveAddressMatch1Flag = LPI2C_SSR_AM1F_MASK,`  
`kLPI2C_SlaveGeneralCallFlag = LPI2C_SSR_GCF_MASK,`  
`kLPI2C_SlaveBusyFlag = LPI2C_SSR_SBF_MASK,`  
`kLPI2C_SlaveBusBusyFlag = LPI2C_SSR_BBF_MASK,`  
`kLPI2C_SlaveClearFlags,`  
`kLPI2C_SlaveIrqFlags,`  
`kLPI2C_SlaveErrorFlags = kLPI2C_SlaveFifoErrFlag | kLPI2C_SlaveBitErrFlag }`  
*LPI2C slave peripheral flags.*
- enum `lpi2c_slave_address_match_t` {  
`kLPI2C_MatchAddress0 = 0U,`  
`kLPI2C_MatchAddress0OrAddress1 = 2U,`  
`kLPI2C_MatchAddress0ThroughAddress1 = 6U }`  
*LPI2C slave address match options.*

- enum `lpi2c_slave_transfer_event_t` {  
`kLPI2C_SlaveAddressMatchEvent` = 0x01U,  
`kLPI2C_SlaveTransmitEvent` = 0x02U,  
`kLPI2C_SlaveReceiveEvent` = 0x04U,  
`kLPI2C_SlaveTransmitAckEvent` = 0x08U,  
`kLPI2C_SlaveRepeatedStartEvent` = 0x10U,  
`kLPI2C_SlaveCompletionEvent` = 0x20U,  
`kLPI2C_SlaveAllEvents` }  
*Set of events sent to the callback for non blocking slave transfers.*

## Slave initialization and deinitialization

- void `LPI2C_SlaveGetDefaultConfig` (`lpi2c_slave_config_t` \*slaveConfig)  
*Provides a default configuration for the LPI2C slave peripheral.*
- void `LPI2C_SlaveInit` (`LPI2C_Type` \*base, const `lpi2c_slave_config_t` \*slaveConfig, `uint32_t` sourceClock\_Hz)  
*Initializes the LPI2C slave peripheral.*
- void `LPI2C_SlaveDeinit` (`LPI2C_Type` \*base)  
*Deinitializes the LPI2C slave peripheral.*
- static void `LPI2C_SlaveReset` (`LPI2C_Type` \*base)  
*Performs a software reset of the LPI2C slave peripheral.*
- static void `LPI2C_SlaveEnable` (`LPI2C_Type` \*base, bool enable)  
*Enables or disables the LPI2C module as slave.*

## Slave status

- static `uint32_t` `LPI2C_SlaveGetStatusFlags` (`LPI2C_Type` \*base)  
*Gets the LPI2C slave status flags.*
- static void `LPI2C_SlaveClearStatusFlags` (`LPI2C_Type` \*base, `uint32_t` statusMask)  
*Clears the LPI2C status flag state.*

## Slave interrupts

- static void `LPI2C_SlaveEnableInterrupts` (`LPI2C_Type` \*base, `uint32_t` interruptMask)  
*Enables the LPI2C slave interrupt requests.*
- static void `LPI2C_SlaveDisableInterrupts` (`LPI2C_Type` \*base, `uint32_t` interruptMask)  
*Disables the LPI2C slave interrupt requests.*
- static `uint32_t` `LPI2C_SlaveGetEnabledInterrupts` (`LPI2C_Type` \*base)  
*Returns the set of currently enabled LPI2C slave interrupt requests.*

## Slave DMA control

- static void `LPI2C_SlaveEnableDMA` (`LPI2C_Type` \*base, bool enableAddressValid, bool enable-Rx, bool enableTx)

*Enables or disables the LPI2C slave peripheral DMA requests.*

## Slave bus operations

- static bool [LPI2C\\_SlaveGetBusIdleState](#) (LPI2C\_Type \*base)  
*Returns whether the bus is idle.*
- static void [LPI2C\\_SlaveTransmitAck](#) (LPI2C\_Type \*base, bool ackOrNack)  
*Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.*
- static void [LPI2C\\_SlaveEnableAckStall](#) (LPI2C\_Type \*base, bool enable)  
*Enables or disables ACKSTALL.*
- static uint32\_t [LPI2C\\_SlaveGetReceivedAddress](#) (LPI2C\_Type \*base)  
*Returns the slave address sent by the I2C master.*
- [status\\_t LPI2C\\_SlaveSend](#) (LPI2C\_Type \*base, void \*txBuff, size\_t txSize, size\_t \*actualTxSize)  
*Performs a polling send transfer on the I2C bus.*
- [status\\_t LPI2C\\_SlaveReceive](#) (LPI2C\_Type \*base, void \*rxBuff, size\_t rxSize, size\_t \*actualRxSize)  
*Performs a polling receive transfer on the I2C bus.*

## Slave non-blocking

- void [LPI2C\\_SlaveTransferCreateHandle](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle, [lpi2c\\_slave\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Creates a new handle for the LPI2C slave non-blocking APIs.*
- [status\\_t LPI2C\\_SlaveTransferNonBlocking](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle, uint32\_t eventMask)  
*Starts accepting slave transfers.*
- [status\\_t LPI2C\\_SlaveTransferGetCount](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle, size\_t \*count)  
*Gets the slave transfer status during a non-blocking transfer.*
- void [LPI2C\\_SlaveTransferAbort](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle)  
*Aborts the slave non-blocking transfers.*

## Slave IRQ handler

- void [LPI2C\\_SlaveTransferHandleIRQ](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle)  
*Reusable routine to handle slave interrupts.*

## 30.5.2 Data Structure Documentation

### 30.5.2.1 struct lpi2c\_slave\_config\_t

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the [LPI2C\\_SlaveGetDefaultConfig\(\)](#) function and pass a pointer to your configuration structure instance.



The configuration structure can be made constant so it resides in flash.

## Data Fields

- bool `enableSlave`  
*Enable slave mode.*
- uint8\_t `address0`  
*Slave's 7-bit address.*
- uint8\_t `address1`  
*Alternate slave 7-bit address.*
- `lpi2c_slave_address_match_t` `addressMatchMode`  
*Address matching options.*
- bool `filterDozeEnable`  
*Enable digital glitch filter in doze mode.*
- bool `filterEnable`  
*Enable digital glitch filter.*
- bool `enableGeneralCall`  
*Enable general call address matching.*
- bool `ignoreAck`  
*Continue transfers after a NACK is detected.*
- bool `enableReceivedAddressRead`  
*Enable reading the address received address as the first byte of data.*
- uint32\_t `sdaGlitchFilterWidth_ns`  
*Width in nanoseconds of the digital filter on the SDA signal.*
- uint32\_t `sclGlitchFilterWidth_ns`  
*Width in nanoseconds of the digital filter on the SCL signal.*
- uint32\_t `dataValidDelay_ns`  
*Width in nanoseconds of the data valid delay.*
- uint32\_t `clockHoldTime_ns`  
*Width in nanoseconds of the clock hold time.*
- bool `enableAck`  
*Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted.*
- bool `enableTx`  
*Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.*
- bool `enableRx`  
*Enables SCL clock stretching when receive data flag is set during a slave-receive transfer.*
- bool `enableAddress`  
*Enables SCL clock stretching when the address valid flag is asserted.*

## Field Documentation

- (1) `bool lpi2c_slave_config_t::enableSlave`
- (2) `uint8_t lpi2c_slave_config_t::address0`
- (3) `uint8_t lpi2c_slave_config_t::address1`
- (4) `lpi2c_slave_address_match_t lpi2c_slave_config_t::addressMatchMode`
- (5) `bool lpi2c_slave_config_t::filterDozeEnable`
- (6) `bool lpi2c_slave_config_t::filterEnable`
- (7) `bool lpi2c_slave_config_t::enableGeneralCall`
- (8) `bool lpi2c_slave_config_t::enableAck`

Clock stretching occurs when transmitting the 9th bit. When `enableAckSCLStall` is enabled, there is no need to set either `enableRxDataSCLStall` or `enableAddressSCLStall`.

- (9) `bool lpi2c_slave_config_t::enableTx`
- (10) `bool lpi2c_slave_config_t::enableRx`
- (11) `bool lpi2c_slave_config_t::enableAddress`
- (12) `bool lpi2c_slave_config_t::ignoreAck`
- (13) `bool lpi2c_slave_config_t::enableReceivedAddressRead`
- (14) `uint32_t lpi2c_slave_config_t::sdaGlitchFilterWidth_ns`

Set to 0 to disable.

- (15) `uint32_t lpi2c_slave_config_t::sclGlitchFilterWidth_ns`

Set to 0 to disable.

- (16) `uint32_t lpi2c_slave_config_t::dataValidDelay_ns`
- (17) `uint32_t lpi2c_slave_config_t::clockHoldTime_ns`

### 30.5.2.2 struct `lpi2c_slave_transfer_t`

#### Data Fields

- `lpi2c_slave_transfer_event_t event`  
*Reason the callback is being invoked.*
- `uint8_t receivedAddress`  
*Matching address send by master.*

- `uint8_t * data`  
*Transfer buffer.*
- `size_t dataSize`  
*Transfer size.*
- `status_t completionStatus`  
*Success or error code describing how the transfer completed.*
- `size_t transferredCount`  
*Number of bytes actually transferred since start or last repeated start.*

### Field Documentation

(1) `lpi2c_slave_transfer_event_t lpi2c_slave_transfer_t::event`

(2) `uint8_t lpi2c_slave_transfer_t::receivedAddress`

(3) `status_t lpi2c_slave_transfer_t::completionStatus`

Only applies for `kLPI2C_SlaveCompletionEvent`.

(4) `size_t lpi2c_slave_transfer_t::transferredCount`

### 30.5.2.3 struct `_lpi2c_slave_handle`

Note

The contents of this structure are private and subject to change.

### Data Fields

- `lpi2c_slave_transfer_t transfer`  
*LPI2C slave transfer copy.*
- `bool isBusy`  
*Whether transfer is busy.*
- `bool wasTransmit`  
*Whether the last transfer was a transmit.*
- `uint32_t eventMask`  
*Mask of enabled events.*
- `uint32_t transferredCount`  
*Count of bytes transferred.*
- `lpi2c_slave_transfer_callback_t callback`  
*Callback function called at transfer event.*
- `void * userData`  
*Callback parameter passed to callback.*

### Field Documentation

- (1) `lpi2c_slave_transfer_t lpi2c_slave_handle_t::transfer`
- (2) `bool lpi2c_slave_handle_t::isBusy`
- (3) `bool lpi2c_slave_handle_t::wasTransmit`
- (4) `uint32_t lpi2c_slave_handle_t::eventMask`
- (5) `uint32_t lpi2c_slave_handle_t::transferredCount`
- (6) `lpi2c_slave_transfer_callback_t lpi2c_slave_handle_t::callback`
- (7) `void* lpi2c_slave_handle_t::userData`

### 30.5.3 Typedef Documentation

**30.5.3.1** `typedef void(* lpi2c_slave_transfer_callback_t)(LPI2C_Type *base, lpi2c_slave_transfer_t *transfer, void *userData)`

This callback is used only for the slave non-blocking transfer API. To install a callback, use the `LPI2C_SlaveSetCallback()` function after you have created a handle.

## Parameters

|                 |                                                                                      |
|-----------------|--------------------------------------------------------------------------------------|
| <i>base</i>     | Base address for the LPI2C instance on which the event occurred.                     |
| <i>transfer</i> | Pointer to transfer descriptor containing values passed to and/or from the callback. |
| <i>userData</i> | Arbitrary pointer-sized value passed from the application.                           |

## 30.5.4 Enumeration Type Documentation

### 30.5.4.1 enum `_lpi2c_slave_flags`

The following status register flags can be cleared:

- [kLPI2C\\_SlaveRepeatedStartDetectFlag](#)
- [kLPI2C\\_SlaveStopDetectFlag](#)
- [kLPI2C\\_SlaveBitErrFlag](#)
- [kLPI2C\\_SlaveFifoErrFlag](#)

All flags except [kLPI2C\\_SlaveBusyFlag](#) and [kLPI2C\\_SlaveBusBusyFlag](#) can be enabled as interrupts.

#### Note

These enumerations are meant to be OR'd together to form a bit mask.

#### Enumerator

- kLPI2C\_SlaveTxReadyFlag*** Transmit data flag.
- kLPI2C\_SlaveRxReadyFlag*** Receive data flag.
- kLPI2C\_SlaveAddressValidFlag*** Address valid flag.
- kLPI2C\_SlaveTransmitAckFlag*** Transmit ACK flag.
- kLPI2C\_SlaveRepeatedStartDetectFlag*** Repeated start detect flag.
- kLPI2C\_SlaveStopDetectFlag*** Stop detect flag.
- kLPI2C\_SlaveBitErrFlag*** Bit error flag.
- kLPI2C\_SlaveFifoErrFlag*** FIFO error flag.
- kLPI2C\_SlaveAddressMatch0Flag*** Address match 0 flag.
- kLPI2C\_SlaveAddressMatch1Flag*** Address match 1 flag.
- kLPI2C\_SlaveGeneralCallFlag*** General call flag.
- kLPI2C\_SlaveBusyFlag*** Master busy flag.
- kLPI2C\_SlaveBusBusyFlag*** Bus busy flag.
- kLPI2C\_SlaveClearFlags*** All flags which are cleared by the driver upon starting a transfer.
- kLPI2C\_SlaveIrqFlags*** IRQ sources enabled by the non-blocking transactional API.
- kLPI2C\_SlaveErrorFlags*** Errors to check for.

### 30.5.4.2 enum lpi2c\_slave\_address\_match\_t

Enumerator

***kLPI2C\_MatchAddress0*** Match only address 0.

***kLPI2C\_MatchAddress0OrAddress1*** Match either address 0 or address 1.

***kLPI2C\_MatchAddress0ThroughAddress1*** Match a range of slave addresses from address 0 through address 1.

### 30.5.4.3 enum lpi2c\_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 [LPI2C\\_SlaveTransferNonBlocking\(\)](#) in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note

These enumerations are meant to be OR'd together to form a bit mask of events.

Enumerator

***kLPI2C\_SlaveAddressMatchEvent*** Received the slave address after a start or repeated start.

***kLPI2C\_SlaveTransmitEvent*** Callback is requested to provide data to transmit (slave-transmitter role).

***kLPI2C\_SlaveReceiveEvent*** Callback is requested to provide a buffer in which to place received data (slave-receiver role).

***kLPI2C\_SlaveTransmitAckEvent*** Callback needs to either transmit an ACK or NACK.

***kLPI2C\_SlaveRepeatedStartEvent*** A repeated start was detected.

***kLPI2C\_SlaveCompletionEvent*** A stop was detected, completing the transfer.

***kLPI2C\_SlaveAllEvents*** Bit mask of all available events.

## 30.5.5 Function Documentation

### 30.5.5.1 void LPI2C\_SlaveGetDefaultConfig ( lpi2c\_slave\_config\_t \* slaveConfig )

This function provides the following default configuration for the LPI2C slave peripheral:

```
* slaveConfig->enableSlave = true;
* slaveConfig->address0 = 0U;
* slaveConfig->address1 = 0U;
* slaveConfig->addressMatchMode = kLPI2C_MatchAddress0;
* slaveConfig->filterDozeEnable = true;
* slaveConfig->filterEnable = true;
* slaveConfig->enableGeneralCall = false;
* slaveConfig->sclStall.enableAck = false;
* slaveConfig->sclStall.enableTx = true;
* slaveConfig->sclStall.enableRx = true;
* slaveConfig->sclStall.enableAddress = true;
```

```

* slaveConfig->ignoreAck = false;
* slaveConfig->enableReceivedAddressRead = false;
* slaveConfig->sdaGlitchFilterWidth_ns = 0;
* slaveConfig->sclGlitchFilterWidth_ns = 0;
* slaveConfig->dataValidDelay_ns = 0;
* slaveConfig->clockHoldTime_ns = 0;
*

```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with [LPI2C\\_SlaveInit\(\)](#). Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

#### Parameters

|     |                    |                                                                                                                      |
|-----|--------------------|----------------------------------------------------------------------------------------------------------------------|
| out | <i>slaveConfig</i> | User provided configuration structure that is set to default values. Refer to <a href="#">lpi2c_slave_config_t</a> . |
|-----|--------------------|----------------------------------------------------------------------------------------------------------------------|

#### 30.5.5.2 void LPI2C\_SlaveInit ( LPI2C\_Type \* *base*, const lpi2c\_slave\_config\_t \* *slaveConfig*, uint32\_t *sourceClock\_Hz* )

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

#### Parameters

|                       |                                                                                                                                           |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>           | The LPI2C peripheral base address.                                                                                                        |
| <i>slaveConfig</i>    | User provided peripheral configuration. Use <a href="#">LPI2C_SlaveGetDefaultConfig()</a> to get a set of defaults that you can override. |
| <i>sourceClock_Hz</i> | Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.             |

#### 30.5.5.3 void LPI2C\_SlaveDeinit ( LPI2C\_Type \* *base* )

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

#### Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

#### 30.5.5.4 static void LPI2C\_SlaveReset ( LPI2C\_Type \* *base* ) [inline], [static]

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

**30.5.5.5** `static void LPI2C_SlaveEnable ( LPI2C_Type * base, bool enable ) [inline], [static]`

Parameters

|               |                                                                       |
|---------------|-----------------------------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                                    |
| <i>enable</i> | Pass true to enable or false to disable the specified LPI2C as slave. |

**30.5.5.6** `static uint32_t LPI2C_SlaveGetStatusFlags ( LPI2C_Type * base ) [inline], [static]`

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

See Also

[\\_lpi2c\\_slave\\_flags](#)

**30.5.5.7** `static void LPI2C_SlaveClearStatusFlags ( LPI2C_Type * base, uint32_t statusMask ) [inline], [static]`

The following status register flags can be cleared:

- [kLPI2C\\_SlaveRepeatedStartDetectFlag](#)
- [kLPI2C\\_SlaveStopDetectFlag](#)
- [kLPI2C\\_SlaveBitErrFlag](#)
- [kLPI2C\\_SlaveFifoErrFlag](#)

Attempts to clear other flags has no effect.



## Parameters

|                   |                                                                                                                                                                                                                                     |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | The LPI2C peripheral base address.                                                                                                                                                                                                  |
| <i>statusMask</i> | A bitmask of status flags that are to be cleared. The mask is composed of <a href="#">_lpi2c_slave_flags</a> enumerators OR'd together. You may pass the result of a previous call to <a href="#">LPI2C_SlaveGetStatusFlags()</a> . |

## See Also

[\\_lpi2c\\_slave\\_flags](#).

### 30.5.5.8 static void LPI2C\_SlaveEnableInterrupts ( LPI2C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

All flags except [kLPI2C\\_SlaveBusyFlag](#) and [kLPI2C\\_SlaveBusBusyFlag](#) can be enabled as interrupts.

## Parameters

|                      |                                                                                                                                                      |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The LPI2C peripheral base address.                                                                                                                   |
| <i>interruptMask</i> | Bit mask of interrupts to enable. See <a href="#">_lpi2c_slave_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

### 30.5.5.9 static void LPI2C\_SlaveDisableInterrupts ( LPI2C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

All flags except [kLPI2C\\_SlaveBusyFlag](#) and [kLPI2C\\_SlaveBusBusyFlag](#) can be enabled as interrupts.

## Parameters

|                      |                                                                                                                                                       |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The LPI2C peripheral base address.                                                                                                                    |
| <i>interruptMask</i> | Bit mask of interrupts to disable. See <a href="#">_lpi2c_slave_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

### 30.5.5.10 static uint32\_t LPI2C\_SlaveGetEnabledInterrupts ( LPI2C\_Type \* *base* ) [inline], [static]

## Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

## Returns

A bitmask composed of `_lpi2c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

**30.5.5.11** `static void LPI2C_SlaveEnableDMA ( LPI2C_Type * base, bool enableAddressValid, bool enableRx, bool enableTx ) [inline], [static]`

## Parameters

|                           |                                                                                                                                                                    |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | The LPI2C peripheral base address.                                                                                                                                 |
| <i>enableAddressValid</i> | Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request. |
| <i>enableRx</i>           | Enable flag for the receive data DMA request. Pass true for enable, false for disable.                                                                             |
| <i>enableTx</i>           | Enable flag for the transmit data DMA request. Pass true for enable, false for disable.                                                                            |

**30.5.5.12** `static bool LPI2C_SlaveGetBusIdleState ( LPI2C_Type * base ) [inline], [static]`

Requires the slave mode to be enabled.

## Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

## Return values

|              |              |
|--------------|--------------|
| <i>true</i>  | Bus is busy. |
| <i>false</i> | Bus is idle. |

**30.5.5.13** `static void LPI2C_SlaveTransmitAck ( LPI2C_Type * base, bool ackOrNack ) [inline], [static]`

Use this function to send an ACK or NAK when the `kLPI2C_SlaveTransmitAckFlag` is asserted. This only happens if you enable the `sclStall.enableAck` field of the `lpi2c_slave_config_t` configuration structure used to initialize the slave peripheral.

Parameters

|                  |                                          |
|------------------|------------------------------------------|
| <i>base</i>      | The LPI2C peripheral base address.       |
| <i>ackOrNack</i> | Pass true for an ACK or false for a NAK. |

#### 30.5.5.14 **static void LPI2C\_SlaveEnableAckStall ( LPI2C\_Type \* *base*, bool *enable* ) [inline], [static]**

When enables ACKSTALL, software can transmit either an ACK or NAK on the I2C bus in response to a byte from the master.

Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                      |
| <i>enable</i> | True will enable ACKSTALL, false will disable ACKSTALL. |

#### 30.5.5.15 **static uint32\_t LPI2C\_SlaveGetReceivedAddress ( LPI2C\_Type \* *base* ) [inline], [static]**

This function should only be called if the [kLPI2C\\_SlaveAddressValidFlag](#) is asserted.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

#### 30.5.5.16 **status\_t LPI2C\_SlaveSend ( LPI2C\_Type \* *base*, void \* *txBuff*, size\_t *txSize*, size\_t \* *actualTxSize* )**

Parameters

|     |                     |                                                    |
|-----|---------------------|----------------------------------------------------|
|     | <i>base</i>         | The LPI2C peripheral base address.                 |
|     | <i>txBuff</i>       | The pointer to the data to be transferred.         |
|     | <i>txSize</i>       | The length in bytes of the data to be transferred. |
| out | <i>actualTxSize</i> |                                                    |

## Returns

Error or success status returned by API.

### 30.5.5.17 `status_t LPI2C_SlaveReceive ( LPI2C_Type * base, void * rxBuff, size_t rxSize, size_t * actualRxSize )`

## Parameters

|     |                     |                                                    |
|-----|---------------------|----------------------------------------------------|
|     | <i>base</i>         | The LPI2C peripheral base address.                 |
|     | <i>rxBuff</i>       | The pointer to the data to be transferred.         |
|     | <i>rxSize</i>       | The length in bytes of the data to be transferred. |
| out | <i>actualRxSize</i> |                                                    |

## Returns

Error or success status returned by API.

### 30.5.5.18 `void LPI2C_SlaveTransferCreateHandle ( LPI2C_Type * base, lpi2c_slave_handle_t * handle, lpi2c_slave_transfer_callback_t callback, void * userData )`

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [LPI2C\\_SlaveTransferAbort\(\)](#) API shall be called.

## Note

The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

## Parameters

|     |                 |                                                              |
|-----|-----------------|--------------------------------------------------------------|
|     | <i>base</i>     | The LPI2C peripheral base address.                           |
| out | <i>handle</i>   | Pointer to the LPI2C slave driver handle.                    |
|     | <i>callback</i> | User provided pointer to the asynchronous callback function. |
|     | <i>userData</i> | User provided pointer to the application callback data.      |

### 30.5.5.19 `status_t LPI2C_SlaveTransferNonBlocking ( LPI2C_Type * base, lpi2c_slave_handle_t * handle, uint32_t eventMask )`

Call this API after calling `I2C_SlaveInit()` and `LPI2C_SlaveTransferCreateHandle()` to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to `LPI2C_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 `lpi2c_slave_transfer_event_t` enumerators for the events you wish to receive. The `kLPI2C_SlaveTransmitEvent` and `kLPI2C_SlaveReceiveEvent` events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the `kLPI2C_SlaveAllEvents` constant is provided as a convenient way to enable all events.

## Parameters

|  |                  |                                                                                                                                                                                                                                                                                                  |
|--|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <i>base</i>      | The LPI2C peripheral base address.                                                                                                                                                                                                                                                               |
|  | <i>handle</i>    | Pointer to <code>lpi2c_slave_handle_t</code> structure which stores the transfer state.                                                                                                                                                                                                          |
|  | <i>eventMask</i> | Bit mask formed by OR'ing together <code>lpi2c_slave_transfer_event_t</code> 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 <code>kLPI2C_SlaveAllEvents</code> to enable all events. |

## Return values

|                           |                                                           |
|---------------------------|-----------------------------------------------------------|
| <i>kStatus_Success</i>    | Slave transfers were successfully started.                |
| <i>kStatus_LPI2C_Busy</i> | Slave transfers have already been started on this handle. |

### 30.5.5.20 `status_t LPI2C_SlaveTransferGetCount ( LPI2C_Type * base, lpi2c_slave_handle_t * handle, size_t * count )`

## Parameters

|     |               |                                                                                                       |
|-----|---------------|-------------------------------------------------------------------------------------------------------|
|     | <i>base</i>   | The LPI2C peripheral base address.                                                                    |
|     | <i>handle</i> | Pointer to <code>i2c_slave_handle_t</code> structure.                                                 |
| out | <i>count</i>  | Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required. |

## Return values

|                                     |  |
|-------------------------------------|--|
| <i>kStatus_Success</i>              |  |
| <i>kStatus_NoTransferInProgress</i> |  |

### 30.5.5.21 void LPI2C\_SlaveTransferAbort ( LPI2C\_Type \* *base*, lpi2c\_slave\_handle\_t \* *handle* )

## Note

This API could be called at any time to stop slave for handling the bus events.

## Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                                                      |
| <i>handle</i> | Pointer to <code>lpi2c_slave_handle_t</code> structure which stores the transfer state. |

## Return values

|                           |  |
|---------------------------|--|
| <i>kStatus_Success</i>    |  |
| <i>kStatus_LPI2C_Idle</i> |  |

### 30.5.5.22 void LPI2C\_SlaveTransferHandleIRQ ( LPI2C\_Type \* *base*, lpi2c\_slave\_handle\_t \* *handle* )

## Note

This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

## Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                                                      |
| <i>handle</i> | Pointer to <code>lpi2c_slave_handle_t</code> structure which stores the transfer state. |

## 30.6 LPI2C Master DMA Driver

### 30.6.1 Overview

#### Data Structures

- struct `lpi2c_master_edma_handle_t`  
*Driver handle for master DMA APIs. [More...](#)*

#### Typedefs

- typedef void(\* `lpi2c_master_edma_transfer_callback_t`)(LPI2C\_Type \*base, lpi2c\_master\_edma\_handle\_t \*handle, `status_t` completionStatus, void \*userData)  
*Master DMA completion callback function pointer type.*

#### Master DMA

- void `LPI2C_MasterCreateEDMAHandle` (LPI2C\_Type \*base, lpi2c\_master\_edma\_handle\_t \*handle, `edma_handle_t` \*rxDmaHandle, `edma_handle_t` \*txDmaHandle, `lpi2c_master_edma_transfer_callback_t` callback, void \*userData)  
*Create a new handle for the LPI2C master DMA APIs.*
- `status_t` `LPI2C_MasterTransferEDMA` (LPI2C\_Type \*base, lpi2c\_master\_edma\_handle\_t \*handle, lpi2c\_master\_transfer\_t \*transfer)  
*Performs a non-blocking DMA-based transaction on the I2C bus.*
- `status_t` `LPI2C_MasterTransferGetCountEDMA` (LPI2C\_Type \*base, lpi2c\_master\_edma\_handle\_t \*handle, `size_t` \*count)  
*Returns number of bytes transferred so far.*
- `status_t` `LPI2C_MasterTransferAbortEDMA` (LPI2C\_Type \*base, lpi2c\_master\_edma\_handle\_t \*handle)  
*Terminates a non-blocking LPI2C master transmission early.*

### 30.6.2 Data Structure Documentation

#### 30.6.2.1 struct `_lpi2c_master_edma_handle`

Note

The contents of this structure are private and subject to change.

#### Data Fields

- LPI2C\_Type \* `base`  
*LPI2C base pointer.*
- bool `isBusy`



- *Transfer state machine current state.*
- `uint8_t nbytes`  
*eDMA minor byte transfer count initially configured.*
- `uint16_t commandBuffer [10]`  
*LPI2C command sequence.*
- `lpi2c_master_transfer_t transfer`  
*Copy of the current transfer info.*
- `lpi2c_master_edma_transfer_callback_t completionCallback`  
*Callback function pointer.*
- `void * userData`  
*Application data passed to callback.*
- `edma_handle_t * rx`  
*Handle for receive DMA channel.*
- `edma_handle_t * tx`  
*Handle for transmit DMA channel.*
- `edma_tcd_t tcds [3]`  
*Software TCD.*

### Field Documentation

- (1) `LPI2C_Type* lpi2c_master_edma_handle_t::base`
- (2) `bool lpi2c_master_edma_handle_t::isBusy`
- (3) `uint8_t lpi2c_master_edma_handle_t::nbytes`
- (4) `uint16_t lpi2c_master_edma_handle_t::commandBuffer[10]`

When all 10 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word] + receive&Size[4 words]

- (5) `lpi2c_master_transfer_t lpi2c_master_edma_handle_t::transfer`
- (6) `lpi2c_master_edma_transfer_callback_t lpi2c_master_edma_handle_t::completionCallback`
- (7) `void* lpi2c_master_edma_handle_t::userData`
- (8) `edma_handle_t* lpi2c_master_edma_handle_t::rx`
- (9) `edma_handle_t* lpi2c_master_edma_handle_t::tx`
- (10) `edma_tcd_t lpi2c_master_edma_handle_t::tcds[3]`

Three are allocated to provide enough room to align to 32-bytes.

### 30.6.3 Typedef Documentation

**30.6.3.1** `typedef void(* lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base, lpi2c_master_edma_handle_t *handle, status_t completionStatus, void *userData)`

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to [LPI2C\\_MasterCreateEDMAHandle\(\)](#).

## Parameters

|                          |                                                                                |
|--------------------------|--------------------------------------------------------------------------------|
| <i>base</i>              | The LPI2C peripheral base address.                                             |
| <i>handle</i>            | Handle associated with the completed transfer.                                 |
| <i>completion-Status</i> | Either kStatus_Success or an error code describing how the transfer completed. |
| <i>userData</i>          | Arbitrary pointer-sized value passed from the application.                     |

### 30.6.4 Function Documentation

**30.6.4.1 void LPI2C\_MasterCreateEDMAHandle ( LPI2C\_Type \* *base*, lpi2c\_master\_edma\_handle\_t \* *handle*, edma\_handle\_t \* *rxDmaHandle*, edma\_handle\_t \* *txDmaHandle*, lpi2c\_master\_edma\_transfer\_callback\_t *callback*, void \* *userData* )**

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [LPI2C\\_MasterTransferAbortEDMA\(\)](#) API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

## Parameters

|     |                    |                                                                                           |
|-----|--------------------|-------------------------------------------------------------------------------------------|
|     | <i>base</i>        | The LPI2C peripheral base address.                                                        |
| out | <i>handle</i>      | Pointer to the LPI2C master driver handle.                                                |
|     | <i>rxDmaHandle</i> | Handle for the eDMA receive channel. Created by the user prior to calling this function.  |
|     | <i>txDmaHandle</i> | Handle for the eDMA transmit channel. Created by the user prior to calling this function. |
|     | <i>callback</i>    | User provided pointer to the asynchronous callback function.                              |
|     | <i>userData</i>    | User provided pointer to the application callback data.                                   |

**30.6.4.2 status\_t LPI2C\_MasterTransferEDMA ( LPI2C\_Type \* *base*, lpi2c\_master\_edma\_handle\_t \* *handle*, lpi2c\_master\_transfer\_t \* *transfer* )**

The callback specified when the *handle* was created is invoked when the transaction has completed.

## Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>base</i>     | The LPI2C peripheral base address.         |
| <i>handle</i>   | Pointer to the LPI2C master driver handle. |
| <i>transfer</i> | The pointer to the transfer descriptor.    |

## Return values

|                           |                                                                                                          |
|---------------------------|----------------------------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>    | The transaction was started successfully.                                                                |
| <i>kStatus_LPI2C_Busy</i> | Either another master is currently utilizing the bus, or another DMA transaction is already in progress. |

### 30.6.4.3 `status_t LPI2C_MasterTransferGetCountEDMA ( LPI2C_Type * base, lpi2c_master_edma_handle_t * handle, size_t * count )`

## Parameters

|     |               |                                                                     |
|-----|---------------|---------------------------------------------------------------------|
|     | <i>base</i>   | The LPI2C peripheral base address.                                  |
|     | <i>handle</i> | Pointer to the LPI2C master driver handle.                          |
| out | <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

## Return values

|                                     |                                                       |
|-------------------------------------|-------------------------------------------------------|
| <i>kStatus_Success</i>              |                                                       |
| <i>kStatus_NoTransferInProgress</i> | There is not a DMA transaction currently in progress. |

### 30.6.4.4 `status_t LPI2C_MasterTransferAbortEDMA ( LPI2C_Type * base, lpi2c_master_edma_handle_t * handle )`

## Note

It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

## Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.         |
| <i>handle</i> | Pointer to the LPI2C master driver handle. |

## Return values

|                           |                                                       |
|---------------------------|-------------------------------------------------------|
| <i>kStatus_Success</i>    | A transaction was successfully aborted.               |
| <i>kStatus_LPI2C_Idle</i> | There is not a DMA transaction currently in progress. |

## 30.7 LPI2C FreeRTOS Driver

### 30.7.1 Overview

#### Driver version

- #define `FSL_LPI2C_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 2)`)  
*LPI2C FreeRTOS driver version.*

#### LPI2C RTOS Operation

- `status_t LPI2C_RTOS_Init` (`lpi2c_rtos_handle_t *handle`, `LPI2C_Type *base`, `const lpi2c_master_config_t *masterConfig`, `uint32_t srcClock_Hz`)  
*Initializes LPI2C.*
- `status_t LPI2C_RTOS_Deinit` (`lpi2c_rtos_handle_t *handle`)  
*Deinitializes the LPI2C.*
- `status_t LPI2C_RTOS_Transfer` (`lpi2c_rtos_handle_t *handle`, `lpi2c_master_transfer_t *transfer`)  
*Performs I2C transfer.*

### 30.7.2 Macro Definition Documentation

#### 30.7.2.1 #define FSL\_LPI2C\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 2))

### 30.7.3 Function Documentation

#### 30.7.3.1 `status_t LPI2C_RTOS_Init ( lpi2c_rtos_handle_t * handle, LPI2C_Type * base, const lpi2c_master_config_t * masterConfig, uint32_t srcClock_Hz )`

This function initializes the LPI2C module and related RTOS context.

Parameters

|                     |                                                                            |
|---------------------|----------------------------------------------------------------------------|
| <i>handle</i>       | The RTOS LPI2C handle, the pointer to an allocated space for RTOS context. |
| <i>base</i>         | The pointer base address of the LPI2C instance to initialize.              |
| <i>masterConfig</i> | Configuration structure to set-up LPI2C in master mode.                    |
| <i>srcClock_Hz</i>  | Frequency of input clock of the LPI2C module.                              |

Returns

status of the operation.

**30.7.3.2 status\_t LPI2C\_RTOS\_Deinit ( lpi2c\_rtos\_handle\_t \* *handle* )**

This function deinitializes the LPI2C module and related RTOS context.

## Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | The RTOS LPI2C handle. |
|---------------|------------------------|

### 30.7.3.3 `status_t LPI2C_RTOS_Transfer ( lpi2c_rtos_handle_t * handle, lpi2c_master_transfer_t * transfer )`

This function performs an I2C transfer using LPI2C module according to data given in the transfer structure.

## Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>handle</i>   | The RTOS LPI2C handle.                        |
| <i>transfer</i> | Structure specifying the transfer parameters. |

## Returns

status of the operation.



## 30.8 LPI2C CMSIS Driver

This section describes the programming interface of the LPI2C 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 LPI2C 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.

### 30.8.1 LPI2C CMSIS Driver

#### 30.8.1.1 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;
```

#### 30.8.1.2 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;
 }
}

/* DMAMux init and EDMA init. */
DMAMUX_Init(EXAMPLE_LPI2C_DMAMUX_BASEADDR);
```

```

edma_config_t edmaConfig;
EDMA_GetDefaultConfig(&edmaConfig);
EDMA_Init(EXAMPLE_LPI2C_DMA_BASEADDR, &edmaConfig);

/*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;

```

### 30.8.1.3 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 31

# LPSPI: Low Power Serial Peripheral Interface

### 31.1 Overview

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

#### Modules

- [LPSPI CMSIS Driver](#)
- [LPSPI FreeRTOS Driver](#)
- [LPSPI Peripheral driver](#)
- [LPSPI eDMA Driver](#)

## 31.2 LPSPI Peripheral driver

### 31.2.1 Overview

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

### 31.2.2 Function groups

#### 31.2.2.1 LPSPI Initialization and De-initialization

This function group initializes the default configuration structure for master and slave, initializes the LPSPI master with a master configuration, initializes the LPSPI slave with a slave configuration, and de-initializes the LPSPI module.

#### 31.2.2.2 LPSPI Basic Operation

This function group enables/disables the LPSPI module both interrupt and DMA, gets the data register address for the DMA transfer, sets master and slave, starts and stops the transfer, and so on.

#### 31.2.2.3 LPSPI Transfer Operation

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

#### 31.2.2.4 LPSPI Status Operation

This function group gets/clears the LPSPI status.

#### 31.2.2.5 LPSPI Block Transfer Operation

This function group transfers a block of data, gets the transfer status, and aborts the transfer.

### 31.2.3 Typical use case

#### 31.2.3.1 Master Operation

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

### 31.2.3.2 Slave Operation

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

#### Data Structures

- struct `lpspi_master_config_t`  
*LPSPI master configuration structure. [More...](#)*
- struct `lpspi_slave_config_t`  
*LPSPI slave configuration structure. [More...](#)*
- struct `lpspi_transfer_t`  
*LPSPI master/slave transfer structure. [More...](#)*
- struct `lpspi_master_handle_t`  
*LPSPI master transfer handle structure used for transactional API. [More...](#)*
- struct `lpspi_slave_handle_t`  
*LPSPI slave transfer handle structure used for transactional API. [More...](#)*

#### Macros

- #define `LPSPI_DUMMY_DATA` (0x00U)  
*LPSPI dummy data if no Tx data.*
- #define `SPI_RETRY_TIMES` 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/  
*Retry times for waiting flag.*
- #define `LPSPI_MASTER_PCS_SHIFT` (4U)  
*LPSPI master PCS shift macro , internal used.*
- #define `LPSPI_MASTER_PCS_MASK` (0xF0U)  
*LPSPI master PCS shift macro , internal used.*
- #define `LPSPI_SLAVE_PCS_SHIFT` (4U)  
*LPSPI slave PCS shift macro , internal used.*
- #define `LPSPI_SLAVE_PCS_MASK` (0xF0U)  
*LPSPI slave PCS shift macro , internal used.*

#### Typedefs

- typedef void(\* `lpspi_master_transfer_callback_t` )(LPSPI\_Type \*base, lpspi\_master\_handle\_t \*handle, `status_t` status, void \*userData)  
*Master completion callback function pointer type.*
- typedef void(\* `lpspi_slave_transfer_callback_t` )(LPSPI\_Type \*base, lpspi\_slave\_handle\_t \*handle, `status_t` status, void \*userData)  
*Slave completion callback function pointer type.*

## Enumerations

- enum {
  - kStatus\_LPSPI\_Busy = MAKE\_STATUS(kStatusGroup\_LPSPI, 0),
  - kStatus\_LPSPI\_Error = MAKE\_STATUS(kStatusGroup\_LPSPI, 1),
  - kStatus\_LPSPI\_Idle = MAKE\_STATUS(kStatusGroup\_LPSPI, 2),
  - kStatus\_LPSPI\_OutOfRange = MAKE\_STATUS(kStatusGroup\_LPSPI, 3),
  - kStatus\_LPSPI\_Timeout = MAKE\_STATUS(kStatusGroup\_LPSPI, 4) }

*Status for the LPSPI driver.*
- enum \_lpspi\_flags {
  - kLPSPI\_TxDataRequestFlag = LPSPI\_SR\_TDF\_MASK,
  - kLPSPI\_RxDataReadyFlag = LPSPI\_SR\_RDF\_MASK,
  - kLPSPI\_WordCompleteFlag = LPSPI\_SR\_WCF\_MASK,
  - kLPSPI\_FrameCompleteFlag = LPSPI\_SR\_FCF\_MASK,
  - kLPSPI\_TransferCompleteFlag = LPSPI\_SR\_TCF\_MASK,
  - kLPSPI\_TransmitErrorFlag = LPSPI\_SR\_TEF\_MASK,
  - kLPSPI\_ReceiveErrorFlag = LPSPI\_SR\_REF\_MASK,
  - kLPSPI\_DataMatchFlag = LPSPI\_SR\_DMF\_MASK,
  - kLPSPI\_ModuleBusyFlag = LPSPI\_SR\_MBF\_MASK,
  - kLPSPI\_AllStatusFlag }

*LPSPI status flags in SPIx\_SR register.*
- enum \_lpspi\_interrupt\_enable {
  - kLPSPI\_TxInterruptEnable = LPSPI\_IER\_TDIE\_MASK,
  - kLPSPI\_RxInterruptEnable = LPSPI\_IER\_RDIE\_MASK,
  - kLPSPI\_WordCompleteInterruptEnable = LPSPI\_IER\_WCIE\_MASK,
  - kLPSPI\_FrameCompleteInterruptEnable = LPSPI\_IER\_FCIE\_MASK,
  - kLPSPI\_TransferCompleteInterruptEnable = LPSPI\_IER\_TCIE\_MASK,
  - kLPSPI\_TransmitErrorInterruptEnable = LPSPI\_IER\_TEIE\_MASK,
  - kLPSPI\_ReceiveErrorInterruptEnable = LPSPI\_IER\_REIE\_MASK,
  - kLPSPI\_DataMatchInterruptEnable = LPSPI\_IER\_DMIE\_MASK,
  - kLPSPI\_AllInterruptEnable }

*LPSPI interrupt source.*
- enum \_lpspi\_dma\_enable {
  - kLPSPI\_TxDmaEnable = LPSPI\_DER\_TDDE\_MASK,
  - kLPSPI\_RxDmaEnable = LPSPI\_DER\_RDDE\_MASK }

*LPSPI DMA source.*
- enum lpspi\_master\_slave\_mode\_t {
  - kLPSPI\_Master = 1U,
  - kLPSPI\_Slave = 0U }

*LPSPI master or slave mode configuration.*
- enum lpspi\_which\_pcs\_t {
  - kLPSPI\_Pcs0 = 0U,
  - kLPSPI\_Pcs1 = 1U,
  - kLPSPI\_Pcs2 = 2U,
  - kLPSPI\_Pcs3 = 3U }

*LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).*

- enum `lpspi_pcs_polarity_config_t` {  
`kLPSPI_PcsActiveHigh = 1U`,  
`kLPSPI_PcsActiveLow = 0U` }  
*LPSPI Peripheral Chip Select (PCS) Polarity configuration.*
- enum `_lpspi_pcs_polarity` {  
`kLPSPI_Pcs0ActiveLow = 1U << 0`,  
`kLPSPI_Pcs1ActiveLow = 1U << 1`,  
`kLPSPI_Pcs2ActiveLow = 1U << 2`,  
`kLPSPI_Pcs3ActiveLow = 1U << 3`,  
`kLPSPI_PcsAllActiveLow = 0xFU` }  
*LPSPI Peripheral Chip Select (PCS) Polarity.*
- enum `lpspi_clock_polarity_t` {  
`kLPSPI_ClockPolarityActiveHigh = 0U`,  
`kLPSPI_ClockPolarityActiveLow = 1U` }  
*LPSPI clock polarity configuration.*
- enum `lpspi_clock_phase_t` {  
`kLPSPI_ClockPhaseFirstEdge = 0U`,  
`kLPSPI_ClockPhaseSecondEdge = 1U` }  
*LPSPI clock phase configuration.*
- enum `lpspi_shift_direction_t` {  
`kLPSPI_MsbFirst = 0U`,  
`kLPSPI_LsbFirst = 1U` }  
*LPSPI data shifter direction options.*
- enum `lpspi_host_request_select_t` {  
`kLPSPI_HostReqExtPin = 0U`,  
`kLPSPI_HostReqInternalTrigger = 1U` }  
*LPSPI Host Request select configuration.*
- enum `lpspi_match_config_t` {  
`kLPSI_MatchDisabled = 0x0U`,  
`kLPSI_1stWordEqualsM0orM1 = 0x2U`,  
`kLPSI_AnyWordEqualsM0orM1 = 0x3U`,  
`kLPSI_1stWordEqualsM0and2ndWordEqualsM1 = 0x4U`,  
`kLPSI_AnyWordEqualsM0andNxtWordEqualsM1 = 0x5U`,  
`kLPSI_1stWordAndM1EqualsM0andM1 = 0x6U`,  
`kLPSI_AnyWordAndM1EqualsM0andM1 = 0x7U` }  
*LPSPI Match configuration options.*
- enum `lpspi_pin_config_t` {  
`kLPSPI_SdiInSdoOut = 0U`,  
`kLPSPI_SdiInSdiOut = 1U`,  
`kLPSPI_SdoInSdoOut = 2U`,  
`kLPSPI_SdoInSdiOut = 3U` }  
*LPSPI pin (SDO and SDI) configuration.*
- enum `lpspi_data_out_config_t` {  
`kLpspiDataOutRetained = 0U`,  
`kLpspiDataOutTristate = 1U` }  
*LPSPI data output configuration.*
- enum `lpspi_transfer_width_t` {

```
kLPSPI_SingleBitXfer = 0U,
kLPSPI_TwoBitXfer = 1U,
kLPSPI_FourBitXfer = 2U }
```

*LPSPI transfer width configuration.*

- enum `lpspi_delay_type_t` {  
`kLPSPI_PcsToSck = 1U,`  
`kLPSPI_LastSckToPcs,`  
`kLPSPI_BetweenTransfer` }

*LPSPI delay type selection.*

- enum `_lpspi_transfer_config_flag_for_master` {  
`kLPSPI_MasterPcs0 = 0U << LPSPI_MASTER_PCS_SHIFT,`  
`kLPSPI_MasterPcs1 = 1U << LPSPI_MASTER_PCS_SHIFT,`  
`kLPSPI_MasterPcs2 = 2U << LPSPI_MASTER_PCS_SHIFT,`  
`kLPSPI_MasterPcs3 = 3U << LPSPI_MASTER_PCS_SHIFT,`  
`kLPSPI_MasterPcsContinuous = 1U << 20,`  
`kLPSPI_MasterByteSwap` }

*Use this enumeration for LPSPI master transfer configFlags.*

- enum `_lpspi_transfer_config_flag_for_slave` {  
`kLPSPI_SlavePcs0 = 0U << LPSPI_SLAVE_PCS_SHIFT,`  
`kLPSPI_SlavePcs1 = 1U << LPSPI_SLAVE_PCS_SHIFT,`  
`kLPSPI_SlavePcs2 = 2U << LPSPI_SLAVE_PCS_SHIFT,`  
`kLPSPI_SlavePcs3 = 3U << LPSPI_SLAVE_PCS_SHIFT,`  
`kLPSPI_SlaveByteSwap` }

*Use this enumeration for LPSPI slave transfer configFlags.*

- enum `_lpspi_transfer_state` {  
`kLPSPI_Idle = 0x0U,`  
`kLPSPI_Busy,`  
`kLPSPI_Error` }

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

## Variables

- volatile `uint8_t g_lpspiDummyData []`  
*Global variable for dummy data value setting.*

## Driver version

- #define `FSL_LPSPI_DRIVER_VERSION (MAKE_VERSION(2, 4, 5))`  
*LPSPI driver version.*

## Initialization and deinitialization

- void `LPSPI_MasterInit` (LPSPI\_Type \*base, const `lpspi_master_config_t` \*masterConfig, `uint32_t` srcClock\_Hz)



- *Initializes the LPSPI master.*
- void [LPSPI\\_MasterGetDefaultConfig](#) ([lpspi\\_master\\_config\\_t](#) \*masterConfig)  
*Sets the [lpspi\\_master\\_config\\_t](#) structure to default values.*
- void [LPSPI\\_SlaveInit](#) ([LPSPI\\_Type](#) \*base, const [lpspi\\_slave\\_config\\_t](#) \*slaveConfig)  
*LPSPI slave configuration.*
- void [LPSPI\\_SlaveGetDefaultConfig](#) ([lpspi\\_slave\\_config\\_t](#) \*slaveConfig)  
*Sets the [lpspi\\_slave\\_config\\_t](#) structure to default values.*
- void [LPSPI\\_Deinit](#) ([LPSPI\\_Type](#) \*base)  
*De-initializes the LPSPI peripheral.*
- void [LPSPI\\_Reset](#) ([LPSPI\\_Type](#) \*base)  
*Restores the LPSPI peripheral to reset state.*
- [uint32\\_t](#) [LPSPI\\_GetInstance](#) ([LPSPI\\_Type](#) \*base)  
*Get the LPSPI instance from peripheral base address.*
- static void [LPSPI\\_Enable](#) ([LPSPI\\_Type](#) \*base, bool enable)  
*Enables the LPSPI peripheral and sets the MCR MDIS to 0.*

## Status

- static [uint32\\_t](#) [LPSPI\\_GetStatusFlags](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI status flag state.*
- static [uint8\\_t](#) [LPSPI\\_GetTxFifoSize](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI Tx FIFO size.*
- static [uint8\\_t](#) [LPSPI\\_GetRxFifoSize](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI Rx FIFO size.*
- static [uint32\\_t](#) [LPSPI\\_GetTxFifoCount](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI Tx FIFO count.*
- static [uint32\\_t](#) [LPSPI\\_GetRxFifoCount](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI Rx FIFO count.*
- static void [LPSPI\\_ClearStatusFlags](#) ([LPSPI\\_Type](#) \*base, [uint32\\_t](#) statusFlags)  
*Clears the LPSPI status flag.*

## Interrupts

- static void [LPSPI\\_EnableInterrupts](#) ([LPSPI\\_Type](#) \*base, [uint32\\_t](#) mask)  
*Enables the LPSPI interrupts.*
- static void [LPSPI\\_DisableInterrupts](#) ([LPSPI\\_Type](#) \*base, [uint32\\_t](#) mask)  
*Disables the LPSPI interrupts.*

## DMA Control

- static void [LPSPI\\_EnableDMA](#) ([LPSPI\\_Type](#) \*base, [uint32\\_t](#) mask)  
*Enables the LPSPI DMA request.*
- static void [LPSPI\\_DisableDMA](#) ([LPSPI\\_Type](#) \*base, [uint32\\_t](#) mask)  
*Disables the LPSPI DMA request.*
- static [uint32\\_t](#) [LPSPI\\_GetTxRegisterAddress](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI Transmit Data Register address for a DMA operation.*
- static [uint32\\_t](#) [LPSPI\\_GetRxRegisterAddress](#) ([LPSPI\\_Type](#) \*base)

*Gets the LPSPI Receive Data Register address for a DMA operation.*

## Bus Operations

- bool [LPSPI\\_CheckTransferArgument](#) (LPSPI\_Type \*base, [lpspi\\_transfer\\_t](#) \*transfer, bool isEdma)  
*Check the argument for transfer .*
- static void [LPSPI\\_SetMasterSlaveMode](#) (LPSPI\_Type \*base, [lpspi\\_master\\_slave\\_mode\\_t](#) mode)  
*Configures the LPSPI for either master or slave.*
- static void [LPSPI\\_SelectTransferPCS](#) (LPSPI\_Type \*base, [lpspi\\_which\\_pcs\\_t](#) select)  
*Configures the peripheral chip select used for the transfer.*
- static void [LPSPI\\_SetPCSContinuous](#) (LPSPI\_Type \*base, bool IsContinuous)  
*Set the PCS signal to continuous or uncontinuous mode.*
- static bool [LPSPI\\_IsMaster](#) (LPSPI\_Type \*base)  
*Returns whether the LPSPI module is in master mode.*
- static void [LPSPI\\_FlushFifo](#) (LPSPI\_Type \*base, bool flushTxFifo, bool flushRxFifo)  
*Flushes the LPSPI FIFOs.*
- static void [LPSPI\\_SetFifoWatermarks](#) (LPSPI\_Type \*base, uint32\_t txWater, uint32\_t rxWater)  
*Sets the transmit and receive FIFO watermark values.*
- static void [LPSPI\\_SetAllPcsPolarity](#) (LPSPI\_Type \*base, uint32\_t mask)  
*Configures all LPSPI peripheral chip select polarities simultaneously.*
- static void [LPSPI\\_SetFrameSize](#) (LPSPI\_Type \*base, uint32\_t frameSize)  
*Configures the frame size.*
- uint32\_t [LPSPI\\_MasterSetBaudRate](#) (LPSPI\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz, uint32\_t \*tcrPrescaleValue)  
*Sets the LPSPI baud rate in bits per second.*
- void [LPSPI\\_MasterSetDelayScaler](#) (LPSPI\_Type \*base, uint32\_t scaler, [lpspi\\_delay\\_type\\_t](#) whichDelay)  
*Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).*
- uint32\_t [LPSPI\\_MasterSetDelayTimes](#) (LPSPI\_Type \*base, uint32\_t delayTimeInNanoSec, [lpspi\\_delay\\_type\\_t](#) whichDelay, uint32\_t srcClock\_Hz)  
*Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).*
- static void [LPSPI\\_WriteData](#) (LPSPI\_Type \*base, uint32\_t data)  
*Writes data into the transmit data buffer.*
- static uint32\_t [LPSPI\\_ReadData](#) (LPSPI\_Type \*base)  
*Reads data from the data buffer.*
- void [LPSPI\\_SetDummyData](#) (LPSPI\_Type \*base, uint8\_t dummyData)  
*Set up the dummy data.*

## Transactional

- void [LPSPI\\_MasterTransferCreateHandle](#) (LPSPI\_Type \*base, [lpspi\\_master\\_handle\\_t](#) \*handle, [lpspi\\_master\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the LPSPI master handle.*
- [status\\_t](#) [LPSPI\\_MasterTransferBlocking](#) (LPSPI\_Type \*base, [lpspi\\_transfer\\_t](#) \*transfer)  
*LPSPi master transfer data using a polling method.*

- [status\\_t LPSPI\\_MasterTransferNonBlocking](#) (LPSPI\_Type \*base, lpspi\_master\_handle\_t \*handle, lpspi\_transfer\_t \*transfer)  
*LPSPI master transfer data using an interrupt method.*
- [status\\_t LPSPI\\_MasterTransferGetCount](#) (LPSPI\_Type \*base, lpspi\_master\_handle\_t \*handle, size\_t \*count)  
*Gets the master transfer remaining bytes.*
- void [LPSPI\\_MasterTransferAbort](#) (LPSPI\_Type \*base, lpspi\_master\_handle\_t \*handle)  
*LPSPI master abort transfer which uses an interrupt method.*
- void [LPSPI\\_MasterTransferHandleIRQ](#) (LPSPI\_Type \*base, lpspi\_master\_handle\_t \*handle)  
*LPSPI Master IRQ handler function.*
- void [LPSPI\\_SlaveTransferCreateHandle](#) (LPSPI\_Type \*base, lpspi\_slave\_handle\_t \*handle, lpspi\_slave\_transfer\_callback\_t callback, void \*userData)  
*Initializes the LPSPI slave handle.*
- [status\\_t LPSPI\\_SlaveTransferNonBlocking](#) (LPSPI\_Type \*base, lpspi\_slave\_handle\_t \*handle, lpspi\_transfer\_t \*transfer)  
*LPSPI slave transfer data using an interrupt method.*
- [status\\_t LPSPI\\_SlaveTransferGetCount](#) (LPSPI\_Type \*base, lpspi\_slave\_handle\_t \*handle, size\_t \*count)  
*Gets the slave transfer remaining bytes.*
- void [LPSPI\\_SlaveTransferAbort](#) (LPSPI\_Type \*base, lpspi\_slave\_handle\_t \*handle)  
*LPSPI slave aborts a transfer which uses an interrupt method.*
- void [LPSPI\\_SlaveTransferHandleIRQ](#) (LPSPI\_Type \*base, lpspi\_slave\_handle\_t \*handle)  
*LPSPI Slave IRQ handler function.*

## 31.2.4 Data Structure Documentation

### 31.2.4.1 struct lpspi\_master\_config\_t

#### Data Fields

- uint32\_t [baudRate](#)  
*Baud Rate for LPSPI.*
- uint32\_t [bitsPerFrame](#)  
*Bits per frame, minimum 8, maximum 4096.*
- [lpspi\\_clock\\_polarity\\_t](#) [cpol](#)  
*Clock polarity.*
- [lpspi\\_clock\\_phase\\_t](#) [cpha](#)  
*Clock phase.*
- [lpspi\\_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](#)  
*Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay.*
- uint32\_t [betweenTransferDelayInNanoSec](#)  
*After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay.*
- [lpspi\\_which\\_pcs\\_t](#) [whichPcs](#)  
*Desired Peripheral Chip Select (PCS).*

- [lpspi\\_pcs\\_polarity\\_config\\_t pcsActiveHighOrLow](#)  
*Desired PCS active high or low.*
- [lpspi\\_pin\\_config\\_t pinCfg](#)  
*Configures which pins are used for input and output data during single bit transfers.*
- [lpspi\\_data\\_out\\_config\\_t dataOutConfig](#)  
*Configures if the output data is tristated between accesses (LPSPI\_PCS is negated).*
- [bool enableInputDelay](#)  
*Enable master to sample the input data on a delayed SCK.*

### Field Documentation

- (1) [uint32\\_t lpspi\\_master\\_config\\_t::baudRate](#)
- (2) [uint32\\_t lpspi\\_master\\_config\\_t::bitsPerFrame](#)
- (3) [lpspi\\_clock\\_polarity\\_t lpspi\\_master\\_config\\_t::cpol](#)
- (4) [lpspi\\_clock\\_phase\\_t lpspi\\_master\\_config\\_t::cpha](#)
- (5) [lpspi\\_shift\\_direction\\_t lpspi\\_master\\_config\\_t::direction](#)
- (6) [uint32\\_t lpspi\\_master\\_config\\_t::pcsToSckDelayInNanoSec](#)

It sets the boundary value if out of range.

- (7) [uint32\\_t lpspi\\_master\\_config\\_t::lastSckToPcsDelayInNanoSec](#)

It sets the boundary value if out of range.

- (8) [uint32\\_t lpspi\\_master\\_config\\_t::betweenTransferDelayInNanoSec](#)

It sets the boundary value if out of range.

- (9) [lpspi\\_which\\_pcs\\_t lpspi\\_master\\_config\\_t::whichPcs](#)
- (10) [lpspi\\_pin\\_config\\_t lpspi\\_master\\_config\\_t::pinCfg](#)
- (11) [lpspi\\_data\\_out\\_config\\_t lpspi\\_master\\_config\\_t::dataOutConfig](#)
- (12) [bool lpspi\\_master\\_config\\_t::enableInputDelay](#)

This can help improve slave setup time. Refer to device data sheet for specific time length.

### 31.2.4.2 struct [lpspi\\_slave\\_config\\_t](#)

#### Data Fields

- [uint32\\_t bitsPerFrame](#)  
*Bits per frame, minimum 8, maximum 4096.*
- [lpspi\\_clock\\_polarity\\_t cpol](#)

- *Clock polarity.*  
`lpspi_clock_phase_t cpha`
- *Clock phase.*  
`lpspi_shift_direction_t direction`
- *MSB or LSB data shift direction.*  
`lpspi_which_pcs_t whichPcs`
- *Desired Peripheral Chip Select (pcs)*  
`lpspi_pcs_polarity_config_t pcsActiveHighOrLow`
- *Desired PCS active high or low.*  
`lpspi_pin_config_t pinCfg`
- *Configures which pins are used for input and output data during single bit transfers.*  
`lpspi_data_out_config_t dataOutConfig`
- *Configures if the output data is tristated between accesses (LPSPI\_PCS is negated).*

### Field Documentation

- (1) `uint32_t lpspi_slave_config_t::bitsPerFrame`
- (2) `lpspi_clock_polarity_t lpspi_slave_config_t::cpol`
- (3) `lpspi_clock_phase_t lpspi_slave_config_t::cpha`
- (4) `lpspi_shift_direction_t lpspi_slave_config_t::direction`
- (5) `lpspi_pin_config_t lpspi_slave_config_t::pinCfg`
- (6) `lpspi_data_out_config_t lpspi_slave_config_t::dataOutConfig`

### 31.2.4.3 struct `lpspi_transfer_t`

#### Data Fields

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

## Field Documentation

- (1) `uint8_t* lpspi_transfer_t::txData`
- (2) `uint8_t* lpspi_transfer_t::rxData`
- (3) `volatile size_t lpspi_transfer_t::dataSize`
- (4) `uint32_t lpspi_transfer_t::configFlags`

Set from `_lpspi_transfer_config_flag_for_master` if the transfer is used for master or `_lpspi_transfer_config_flag_for_slave` enumeration if the transfer is used for slave.

### 31.2.4.4 struct `_lpspi_master_handle`

Forward declaration of the `_lpspi_master_handle` typedefs.

## Data Fields

- volatile bool `isPcsContinuous`  
*Is PCS continuous in transfer.*
- volatile bool `writeTcrInIsr`  
*A flag that whether should write TCR in ISR.*
- volatile bool `isByteSwap`  
*A flag that whether should byte swap.*
- volatile bool `isTxMask`  
*A flag that whether TCR[TXMSK] is set.*
- volatile `uint16_t bytesPerFrame`  
*Number of bytes in each frame.*
- volatile `uint8_t fifoSize`  
*FIFO dataSize.*
- volatile `uint8_t rxWatermark`  
*Rx watermark.*
- volatile `uint8_t bytesEachWrite`  
*Bytes for each write TDR.*
- volatile `uint8_t bytesEachRead`  
*Bytes for each read RDR.*
- `uint8_t *volatile txData`  
*Send buffer.*
- `uint8_t *volatile rxData`  
*Receive buffer.*
- volatile `size_t txRemainingByteCount`  
*Number of bytes remaining to send.*
- volatile `size_t rxRemainingByteCount`  
*Number of bytes remaining to receive.*
- volatile `uint32_t writeRegRemainingTimes`  
*Write TDR register remaining times.*
- volatile `uint32_t readRegRemainingTimes`  
*Read RDR register remaining times.*

- `uint32_t totalByteCount`  
*Number of transfer bytes.*
- `uint32_t txBuffIfNull`  
*Used if the txData is NULL.*
- `volatile uint8_t state`  
*LPSPI transfer state , `_lpspi_transfer_state`.*
- `lpspi_master_transfer_callback_t callback`  
*Completion callback.*
- `void * userData`  
*Callback user data.*

## Field Documentation

- (1) `volatile bool lpspi_master_handle_t::isPcsContinuous`
- (2) `volatile bool lpspi_master_handle_t::writeTcrInlSr`
- (3) `volatile bool lpspi_master_handle_t::isByteSwap`
- (4) `volatile bool lpspi_master_handle_t::isTxMask`
- (5) `volatile uint8_t lpspi_master_handle_t::fifoSize`
- (6) `volatile uint8_t lpspi_master_handle_t::rxWatermark`
- (7) `volatile uint8_t lpspi_master_handle_t::bytesEachWrite`
- (8) `volatile uint8_t lpspi_master_handle_t::bytesEachRead`
- (9) `uint8_t* volatile lpspi_master_handle_t::txData`
- (10) `uint8_t* volatile lpspi_master_handle_t::rxData`
- (11) `volatile size_t lpspi_master_handle_t::txRemainingByteCount`
- (12) `volatile size_t lpspi_master_handle_t::rxRemainingByteCount`
- (13) `volatile uint32_t lpspi_master_handle_t::writeRegRemainingTimes`
- (14) `volatile uint32_t lpspi_master_handle_t::readRegRemainingTimes`
- (15) `uint32_t lpspi_master_handle_t::txBuffIfNull`
- (16) `volatile uint8_t lpspi_master_handle_t::state`
- (17) `lpspi_master_transfer_callback_t lpspi_master_handle_t::callback`
- (18) `void* lpspi_master_handle_t::userData`

### 31.2.4.5 struct `_lpspi_slave_handle`

Forward declaration of the `_lpspi_slave_handle` typedefs.

## Data Fields

- `volatile bool isByteSwap`  
*A flag that whether should byte swap.*
- `volatile uint8_t fifoSize`  
*FIFO dataSize.*
- `volatile uint8_t rxWatermark`  
*Rx watermark.*



- volatile uint8\_t `bytesEachWrite`  
*Bytes for each write TDR.*
- volatile uint8\_t `bytesEachRead`  
*Bytes for each read RDR.*
- uint8\_t \*volatile `txData`  
*Send buffer.*
- uint8\_t \*volatile `rxData`  
*Receive buffer.*
- volatile size\_t `txRemainingByteCount`  
*Number of bytes remaining to send.*
- volatile size\_t `rxRemainingByteCount`  
*Number of bytes remaining to receive.*
- volatile uint32\_t `writeRegRemainingTimes`  
*Write TDR register remaining times.*
- volatile uint32\_t `readRegRemainingTimes`  
*Read RDR register remaining times.*
- uint32\_t `totalByteCount`  
*Number of transfer bytes.*
- volatile uint8\_t `state`  
*LPSPI transfer state , `_lpspi_transfer_state`.*
- volatile uint32\_t `errorCount`  
*Error count for slave transfer.*
- `lpspi_slave_transfer_callback_t` `callback`  
*Completion callback.*
- void \* `userData`  
*Callback user data.*

**Field Documentation**

- (1) `volatile bool lpspi_slave_handle_t::isByteSwap`
- (2) `volatile uint8_t lpspi_slave_handle_t::fifoSize`
- (3) `volatile uint8_t lpspi_slave_handle_t::rxWatermark`
- (4) `volatile uint8_t lpspi_slave_handle_t::bytesEachWrite`
- (5) `volatile uint8_t lpspi_slave_handle_t::bytesEachRead`
- (6) `uint8_t* volatile lpspi_slave_handle_t::txData`
- (7) `uint8_t* volatile lpspi_slave_handle_t::rxData`
- (8) `volatile size_t lpspi_slave_handle_t::txRemainingByteCount`
- (9) `volatile size_t lpspi_slave_handle_t::rxRemainingByteCount`
- (10) `volatile uint32_t lpspi_slave_handle_t::writeRegRemainingTimes`
- (11) `volatile uint32_t lpspi_slave_handle_t::readRegRemainingTimes`
- (12) `volatile uint8_t lpspi_slave_handle_t::state`
- (13) `volatile uint32_t lpspi_slave_handle_t::errorCount`
- (14) `lpspi_slave_transfer_callback_t lpspi_slave_handle_t::callback`
- (15) `void* lpspi_slave_handle_t::userData`

**31.2.5 Macro Definition Documentation**

**31.2.5.1 #define FSL\_LPSPI\_DRIVER\_VERSION (MAKE\_VERSION(2, 4, 5))**

**31.2.5.2 #define LPSPI\_DUMMY\_DATA (0x00U)**

Dummy data used for tx if there is not txData.

**31.2.5.3** `#define SPI_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`

**31.2.5.4** `#define LPSPI_MASTER_PCS_SHIFT (4U)`

**31.2.5.5** `#define LPSPI_MASTER_PCS_MASK (0xF0U)`

**31.2.5.6** `#define LPSPI_SLAVE_PCS_SHIFT (4U)`

**31.2.5.7** `#define LPSPI_SLAVE_PCS_MASK (0xF0U)`

## **31.2.6 Typedef Documentation**

**31.2.6.1** `typedef void(* lpspi_master_transfer_callback_t)(LPSPI_Type *base, lpspi_master_handle_t *handle, status_t status, void *userData)`

## Parameters

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

### 31.2.6.2 typedef void(\* lpspi\_slave\_transfer\_callback\_t)(LPSPI\_Type \*base, lpspi\_slave\_handle\_t \*handle, status\_t status, void \*userData)

## Parameters

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

## 31.2.7 Enumeration Type Documentation

### 31.2.7.1 anonymous enum

## Enumerator

- kStatus\_LPSPI\_Busy* LPSPI transfer is busy.
- kStatus\_LPSPI\_Error* LPSPI driver error.
- kStatus\_LPSPI\_Idle* LPSPI is idle.
- kStatus\_LPSPI\_OutOfRange* LPSPI transfer out Of range.
- kStatus\_LPSPI\_Timeout* LPSPI timeout polling status flags.

### 31.2.7.2 enum \_lpspi\_flags

## Enumerator

- kLPSPI\_TxDataRequestFlag* Transmit data flag.
- kLPSPI\_RxDataReadyFlag* Receive data flag.
- kLPSPI\_WordCompleteFlag* Word Complete flag.
- kLPSPI\_FrameCompleteFlag* Frame Complete flag.
- kLPSPI\_TransferCompleteFlag* Transfer Complete flag.
- kLPSPI\_TransmitErrorFlag* Transmit Error flag (FIFO underrun)
- kLPSPI\_ReceiveErrorFlag* Receive Error flag (FIFO overrun)

*kLPSPI\_DataMatchFlag* Data Match flag.  
*kLPSPI\_ModuleBusyFlag* Module Busy flag.  
*kLPSPI\_AllStatusFlag* Used for clearing all w1c status flags.

### 31.2.7.3 enum \_lpspi\_interrupt\_enable

Enumerator

*kLPSPI\_TxInterruptEnable* Transmit data interrupt enable.  
*kLPSPI\_RxInterruptEnable* Receive data interrupt enable.  
*kLPSPI\_WordCompleteInterruptEnable* Word complete interrupt enable.  
*kLPSPI\_FrameCompleteInterruptEnable* Frame complete interrupt enable.  
*kLPSPI\_TransferCompleteInterruptEnable* Transfer complete interrupt enable.  
*kLPSPI\_TransmitErrorInterruptEnable* Transmit error interrupt enable(FIFO underrun)  
*kLPSPI\_ReceiveErrorInterruptEnable* Receive Error interrupt enable (FIFO overrun)  
*kLPSPI\_DataMatchInterruptEnable* Data Match interrupt enable.  
*kLPSPI\_AllInterruptEnable* All above interrupts enable.

### 31.2.7.4 enum \_lpspi\_dma\_enable

Enumerator

*kLPSPI\_TxDmaEnable* Transmit data DMA enable.  
*kLPSPI\_RxDmaEnable* Receive data DMA enable.

### 31.2.7.5 enum lpspi\_master\_slave\_mode\_t

Enumerator

*kLPSPI\_Master* LPSPI peripheral operates in master mode.  
*kLPSPI\_Slave* LPSPI peripheral operates in slave mode.

### 31.2.7.6 enum lpspi\_which\_pcs\_t

Enumerator

*kLPSPI\_Pcs0* PCS[0].  
*kLPSPI\_Pcs1* PCS[1].  
*kLPSPI\_Pcs2* PCS[2].  
*kLPSPI\_Pcs3* PCS[3].

**31.2.7.7 enum lpspi\_pcs\_polarity\_config\_t**

Enumerator

*kLPSPI\_PcsActiveHigh* PCS Active High (idles low)  
*kLPSPI\_PcsActiveLow* PCS Active Low (idles high)

**31.2.7.8 enum \_lpspi\_pcs\_polarity**

Enumerator

*kLPSPI\_Pcs0ActiveLow* Pcs0 Active Low (idles high).  
*kLPSPI\_Pcs1ActiveLow* Pcs1 Active Low (idles high).  
*kLPSPI\_Pcs2ActiveLow* Pcs2 Active Low (idles high).  
*kLPSPI\_Pcs3ActiveLow* Pcs3 Active Low (idles high).  
*kLPSPI\_PcsAllActiveLow* Pcs0 to Pcs5 Active Low (idles high).

**31.2.7.9 enum lpspi\_clock\_polarity\_t**

Enumerator

*kLPSPI\_ClockPolarityActiveHigh* CPOL=0. Active-high LPSPI clock (idles low)  
*kLPSPI\_ClockPolarityActiveLow* CPOL=1. Active-low LPSPI clock (idles high)

**31.2.7.10 enum lpspi\_clock\_phase\_t**

Enumerator

*kLPSPI\_ClockPhaseFirstEdge* CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.  
*kLPSPI\_ClockPhaseSecondEdge* CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

**31.2.7.11 enum lpspi\_shift\_direction\_t**

Enumerator

*kLPSPI\_MsbFirst* Data transfers start with most significant bit.  
*kLPSPI\_LsbFirst* Data transfers start with least significant bit.

**31.2.7.12 enum lpspi\_host\_request\_select\_t**

Enumerator

*kLPSPI\_HostReqExtPin* Host Request is an ext pin.*kLPSPI\_HostReqInternalTrigger* Host Request is an internal trigger.**31.2.7.13 enum lpspi\_match\_config\_t**

Enumerator

*kLPSI\_MatchDisabled* LPSPI Match Disabled.*kLPSI\_1stWordEqualsM0orM1* LPSPI Match Enabled.*kLPSI\_AnyWordEqualsM0orM1* LPSPI Match Enabled.*kLPSI\_1stWordEqualsM0and2ndWordEqualsM1* LPSPI Match Enabled.*kLPSI\_AnyWordEqualsM0andNxtWordEqualsM1* LPSPI Match Enabled.*kLPSI\_1stWordAndM1EqualsM0andM1* LPSPI Match Enabled.*kLPSI\_AnyWordAndM1EqualsM0andM1* LPSPI Match Enabled.**31.2.7.14 enum lpspi\_pin\_config\_t**

Enumerator

*kLPSPI\_SdiInSdoOut* LPSPI SDI input, SDO output.*kLPSPI\_SdiInSdiOut* LPSPI SDI input, SDI output.*kLPSPI\_SdoInSdoOut* LPSPI SDO input, SDO output.*kLPSPI\_SdoInSdiOut* LPSPI SDO input, SDI output.**31.2.7.15 enum lpspi\_data\_out\_config\_t**

Enumerator

*kLpspiDataOutRetained* Data out retains last value when chip select is de-asserted.*kLpspiDataOutTristate* Data out is tristated when chip select is de-asserted.**31.2.7.16 enum lpspi\_transfer\_width\_t**

Enumerator

*kLPSPI\_SingleBitXfer* 1-bit shift at a time, data out on SDO, in on SDI (normal mode)*kLPSPI\_TwoBitXfer* 2-bits shift out on SDO/SDI and in on SDO/SDI*kLPSPI\_FourBitXfer* 4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

### 31.2.7.17 enum `lpspi_delay_type_t`

Enumerator

- kLPSPI\_PcsToSck* PCS-to-SCK delay.
- kLPSPI\_LastSckToPcs* Last SCK edge to PCS delay.
- kLPSPI\_BetweenTransfer* Delay between transfers.

### 31.2.7.18 enum `_lpspi_transfer_config_flag_for_master`

Enumerator

- kLPSPI\_MasterPcs0* LPSPI master transfer use PCS0 signal.
- kLPSPI\_MasterPcs1* LPSPI master transfer use PCS1 signal.
- kLPSPI\_MasterPcs2* LPSPI master transfer use PCS2 signal.
- kLPSPI\_MasterPcs3* LPSPI master transfer use PCS3 signal.
- kLPSPI\_MasterPcsContinuous* Is PCS signal continuous.
- kLPSPI\_MasterByteSwap* Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set `lpspi_shift_direction_t` to MSB).
  1. If you set `bitPerFrame = 8`, no matter the `kLPSPI_MasterByteSwap` flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
  2. If you set `bitPerFrame = 16`: (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the `kLPSPI_MasterByteSwap` flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the `kLPSPI_MasterByteSwap` flag.
  3. If you set `bitPerFrame = 32`: (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the `kLPSPI_MasterByteSwap` flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the `kLPSPI_MasterByteSwap` flag.

### 31.2.7.19 enum `_lpspi_transfer_config_flag_for_slave`

Enumerator

- kLPSPI\_SlavePcs0* LPSPI slave transfer use PCS0 signal.
- kLPSPI\_SlavePcs1* LPSPI slave transfer use PCS1 signal.
- kLPSPI\_SlavePcs2* LPSPI slave transfer use PCS2 signal.
- kLPSPI\_SlavePcs3* LPSPI slave transfer use PCS3 signal.
- kLPSPI\_SlaveByteSwap* Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set `lpspi_shift_direction_t` to MSB).
  1. If you set `bitPerFrame = 8`, no matter the `kLPSPI_SlaveByteSwap` flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
  2. If you set `bitPerFrame = 16`: (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the `kLPSPI_SlaveByteSwap` flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the `kLPSPI_SlaveByteSwap` flag.



- If you set `bitPerFrame = 32` : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the `kLPSPi_SlaveByteSwap` flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the `kLPSPi_SlaveByteSwap` flag.

### 31.2.7.20 enum \_lpspi\_transfer\_state

Enumerator

- kLPSPi\_Idle* Nothing in the transmitter/receiver.
- kLPSPi\_Busy* Transfer queue is not finished.
- kLPSPi\_Error* Transfer error.

## 31.2.8 Function Documentation

### 31.2.8.1 void LPSPi\_MasterInit ( LPSPi\_Type \* base, const lpspi\_master\_config\_t \* masterConfig, uint32\_t srcClock\_Hz )

Parameters

|                     |                                                              |
|---------------------|--------------------------------------------------------------|
| <i>base</i>         | LPSPi peripheral address.                                    |
| <i>masterConfig</i> | Pointer to structure <a href="#">lpspi_master_config_t</a> . |
| <i>srcClock_Hz</i>  | Module source input clock in Hertz                           |

### 31.2.8.2 void LPSPi\_MasterGetDefaultConfig ( lpspi\_master\_config\_t \* masterConfig )

This API initializes the configuration structure for [LPSPi\\_MasterInit\(\)](#). The initialized structure can remain unchanged in [LPSPi\\_MasterInit\(\)](#), or can be modified before calling the [LPSPi\\_MasterInit\(\)](#). Example:

```
* lpspi_master_config_t masterConfig;
* LPSPi_MasterGetDefaultConfig(&masterConfig);
*
```

Parameters

|                     |                                                            |
|---------------------|------------------------------------------------------------|
| <i>masterConfig</i> | pointer to <a href="#">lpspi_master_config_t</a> structure |
|---------------------|------------------------------------------------------------|

### 31.2.8.3 void LPSPi\_SlaveInit ( LPSPi\_Type \* base, const lpspi\_slave\_config\_t \* slaveConfig )

Parameters

|                    |                                                               |
|--------------------|---------------------------------------------------------------|
| <i>base</i>        | LPSPI peripheral address.                                     |
| <i>slaveConfig</i> | Pointer to a structure <a href="#">lpspi_slave_config_t</a> . |

#### 31.2.8.4 void LPSPI\_SlaveGetDefaultConfig ( lpspi\_slave\_config\_t \* slaveConfig )

This API initializes the configuration structure for [LPSPI\\_SlaveInit\(\)](#). The initialized structure can remain unchanged in [LPSPI\\_SlaveInit\(\)](#) or can be modified before calling the [LPSPI\\_SlaveInit\(\)](#). Example:

```
* lpspi_slave_config_t slaveConfig;
* LPSPI_SlaveGetDefaultConfig(&slaveConfig);
*
```

Parameters

|                    |                                                            |
|--------------------|------------------------------------------------------------|
| <i>slaveConfig</i> | pointer to <a href="#">lpspi_slave_config_t</a> structure. |
|--------------------|------------------------------------------------------------|

#### 31.2.8.5 void LPSPI\_Deinit ( LPSPI\_Type \* base )

Call this API to disable the LPSPI clock.

Parameters

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

#### 31.2.8.6 void LPSPI\_Reset ( LPSPI\_Type \* base )

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

Parameters

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

#### 31.2.8.7 uint32\_t LPSPI\_GetInstance ( LPSPI\_Type \* base )

## Parameters

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

## Returns

LPSPI instance.

**31.2.8.8** `static void LPSPI_Enable ( LPSPI_Type * base, bool enable ) [inline],  
[static]`

## Parameters

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral address.                            |
| <i>enable</i> | Pass true to enable module, false to disable module. |

**31.2.8.9** `static uint32_t LPSPI_GetStatusFlags ( LPSPI_Type * base ) [inline],  
[static]`

## Parameters

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

## Returns

The LPSPI status(in SR register).

**31.2.8.10** `static uint8_t LPSPI_GetTxFifoSize ( LPSPI_Type * base ) [inline],  
[static]`

## Parameters

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

## Returns

The LPSPI Tx FIFO size.

**31.2.8.11** `static uint8_t LPSPI_GetRxFifoSize ( LPSPI_Type * base ) [inline],  
[static]`

## Parameters

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

## Returns

The LPSPI Rx FIFO size.

**31.2.8.12** `static uint32_t LPSPI_GetTxFifoCount ( LPSPI_Type * base ) [inline], [static]`

## Parameters

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

## Returns

The number of words in the transmit FIFO.

**31.2.8.13** `static uint32_t LPSPI_GetRxFifoCount ( LPSPI_Type * base ) [inline], [static]`

## Parameters

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

## Returns

The number of words in the receive FIFO.

**31.2.8.14** `static void LPSPI_ClearStatusFlags ( LPSPI_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 flag bit to clear. The list of status flags is defined in the `_lpspi_flags`. Example usage:

```
* LPSPI_ClearStatusFlags(base, kLPSPI_TxDataRequestFlag |
* kLPSPI_RxDataReadyFlag);
```

\*

Parameters

|                    |                                                            |
|--------------------|------------------------------------------------------------|
| <i>base</i>        | LPSPI peripheral address.                                  |
| <i>statusFlags</i> | The status flag used from type <code>_lpspi_flags</code> . |

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

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

This function configures the various interrupt masks of the LPSPI. The parameters are base and an interrupt mask. Note that, for Tx fill and Rx FIFO drain requests, enabling the interrupt request disables the DMA request.

```
* LPSPI_EnableInterrupts(base, kLPSPI_TxInterruptEnable |
 kLPSPI_RxInterruptEnable);
*
```

Parameters

|             |                                                                         |
|-------------|-------------------------------------------------------------------------|
| <i>base</i> | LPSPI peripheral address.                                               |
| <i>mask</i> | The interrupt mask; Use the enum <code>_lpspi_interrupt_enable</code> . |

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

```
* LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable |
 kLPSPI_RxInterruptEnable);
*
```

Parameters

|             |                                                                         |
|-------------|-------------------------------------------------------------------------|
| <i>base</i> | LPSPI peripheral address.                                               |
| <i>mask</i> | The interrupt mask; Use the enum <code>_lpspi_interrupt_enable</code> . |

**31.2.8.17 static void LPSPI\_EnableDMA ( LPSPI\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
* LPSPI_EnableDMA(base, kLPSPI_TxDmaEnable |
 kLPSPI_RxDmaEnable);
*
```

## Parameters

|             |                                                                   |
|-------------|-------------------------------------------------------------------|
| <i>base</i> | LPSPI peripheral address.                                         |
| <i>mask</i> | The interrupt mask; Use the enum <code>_lpspi_dma_enable</code> . |

### 31.2.8.18 `static void LPSPI_DisableDMA ( LPSPI_Type * base, uint32_t mask )` `[inline], [static]`

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are *base* and a DMA mask.

```
* SPI_DisableDMA(base, kLPSPI_TxDmaEnable |
 kLPSPI_RxDmaEnable);
*
```

## Parameters

|             |                                                                   |
|-------------|-------------------------------------------------------------------|
| <i>base</i> | LPSPI peripheral address.                                         |
| <i>mask</i> | The interrupt mask; Use the enum <code>_lpspi_dma_enable</code> . |

### 31.2.8.19 `static uint32_t LPSPI_GetTxRegisterAddress ( LPSPI_Type * base )` `[inline], [static]`

This function gets the LPSPI Transmit Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

## Parameters

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

## Returns

The LPSPI Transmit Data Register address.

### 31.2.8.20 `static uint32_t LPSPI_GetRxRegisterAddress ( LPSPI_Type * base )` `[inline], [static]`

This function gets the LPSPI Receive Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

## Parameters

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

## Returns

The LPSPI Receive Data Register address.

### 31.2.8.21 **bool LPSPI\_CheckTransferArgument ( LPSPI\_Type \* *base*, lpspi\_transfer\_t \* *transfer*, bool *isEdma* )**

## Parameters

|                 |                                                                                 |
|-----------------|---------------------------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral address.                                                       |
| <i>transfer</i> | the transfer struct to be used.                                                 |
| <i>isEdma</i>   | True to check for EDMA transfer, false to check interrupt non-blocking transfer |

## Returns

Return true for right and false for wrong.

### 31.2.8.22 **static void LPSPI\_SetMasterSlaveMode ( LPSPI\_Type \* *base*, lpspi\_master\_slave\_mode\_t *mode* ) [inline], [static]**

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx\_CR\_MEN = 0).

## Parameters

|             |                                                                   |
|-------------|-------------------------------------------------------------------|
| <i>base</i> | LPSPI peripheral address.                                         |
| <i>mode</i> | Mode setting (master or slave) of type lpspi_master_slave_mode_t. |

### 31.2.8.23 **static void LPSPI\_SelectTransferPCS ( LPSPI\_Type \* *base*, lpspi\_which\_pcs\_t *select* ) [inline], [static]**

Parameters

|               |                                                   |
|---------------|---------------------------------------------------|
| <i>base</i>   | LPSPI peripheral address.                         |
| <i>select</i> | LPSPI Peripheral Chip Select (PCS) configuration. |

**31.2.8.24 static void LPSPI\_SetPCSContinuous ( LPSPI\_Type \* *base*, bool *IsContinuous* ) [inline], [static]**

Note

In master mode, continuous transfer will keep the PCS asserted at the end of the frame size, until a command word is received that starts a new frame. So PCS must be set back to uncontinuous when transfer finishes. In slave mode, when continuous transfer is enabled, the LPSPI will only transmit the first frame size bits, after that the LPSPI will transmit received data back (assuming a 32-bit shift register).

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>base</i>         | LPSPI peripheral address.                                                           |
| <i>IsContinuous</i> | True to set the transfer PCS to continuous mode, false to set to uncontinuous mode. |

**31.2.8.25 static bool LPSPI\_IsMaster ( LPSPI\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

**31.2.8.26 static void LPSPI\_FlushFifo ( LPSPI\_Type \* *base*, bool *flushTxFifo*, bool *flushRxFifo* ) [inline], [static]**

Parameters



|                    |                                                                    |
|--------------------|--------------------------------------------------------------------|
| <i>base</i>        | LPSPI peripheral address.                                          |
| <i>flushTxFifo</i> | Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO. |
| <i>flushRxFifo</i> | Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO. |

### 31.2.8.27 static void LPSPI\_SetFifoWatermarks ( LPSPI\_Type \* *base*, uint32\_t *txWater*, uint32\_t *rxWater* ) [inline], [static]

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

Parameters

|                |                                                                                                |
|----------------|------------------------------------------------------------------------------------------------|
| <i>base</i>    | LPSPI peripheral address.                                                                      |
| <i>txWater</i> | The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated. |
| <i>rxWater</i> | The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated. |

### 31.2.8.28 static void LPSPI\_SetAllPcsPolarity ( LPSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx\_CR\_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
* LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow |
 kLPSPI_Pcs1ActiveLow);
*
```

Parameters

|             |                                                                        |
|-------------|------------------------------------------------------------------------|
| <i>base</i> | LPSPI peripheral address.                                              |
| <i>mask</i> | The PCS polarity mask; Use the enum <code>_lpspi_pcs_polarity</code> . |

### 31.2.8.29 static void LPSPI\_SetFrameSize ( LPSPI\_Type \* *base*, uint32\_t *frameSize* ) [inline], [static]

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame size is greater than 32-bits, the

word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPI in slave mode, although the command register does not update until after the LPSPI is enabled. After it is enabled, the transmit command register should only be changed if the LPSPI is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

Parameters

|                  |                                   |
|------------------|-----------------------------------|
| <i>base</i>      | LPSPI peripheral address.         |
| <i>frameSize</i> | The frame size in number of bits. |

### 31.2.8.30 `uint32_t LPSPI_MasterSetBaudRate ( LPSPI_Type * base, uint32_t baudRate_Bps, uint32_t srcClock_Hz, uint32_t * tcrPrescaleValue )`

This function takes in the desired bitsPerSec (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 the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale `tcrPrescaleValue` parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

|                          |                                                   |
|--------------------------|---------------------------------------------------|
| <i>base</i>              | LPSPI peripheral address.                         |
| <i>baudRate_Bps</i>      | The desired baud rate in bits per second.         |
| <i>srcClock_Hz</i>       | Module source input clock in Hertz.               |
| <i>tcrPrescale-Value</i> | The TCR prescale value needed to program the TCR. |

## Returns

The actual calculated baud rate. This function may also return a "0" if the LPSPI is not configured for master mode or if the LPSPI module is not disabled.

### 31.2.8.31 void LPSPI\_MasterSetDelayScaler ( LPSPI\_Type \* *base*, uint32\_t *scaler*, lpspi\_delay\_type\_t *whichDelay* )

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type lpspi\_delay\_type\_t.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

## Parameters

|                   |                                                                     |
|-------------------|---------------------------------------------------------------------|
| <i>base</i>       | LPSPI peripheral address.                                           |
| <i>scaler</i>     | The 8-bit delay value 0x00 to 0xFF (255).                           |
| <i>whichDelay</i> | The desired delay to configure, must be of type lpspi_delay_type_t. |

### 31.2.8.32 uint32\_t LPSPI\_MasterSetDelayTimes ( LPSPI\_Type \* *base*, uint32\_t *delayTimeInNanoSec*, lpspi\_delay\_type\_t *whichDelay*, uint32\_t *srcClock\_Hz* )

This function calculates the values for the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type lpspi\_delay\_type\_t.

The user passes the desired delay and the desired delay value in nano-seconds. The function calculates the value needed for the desired delay parameter and returns the actual calculated delay because 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. It is up to the higher level peripheral driver to alert the user of an out of range delay input.

Note that the LPSPI module must be configured for master mode before configuring this. And note that the  $\text{delayTime} = \text{LPSPI\_clockSource} / (\text{PRESCALE} * \text{Delay\_scaler})$ .

Parameters

|                           |                                                                                             |
|---------------------------|---------------------------------------------------------------------------------------------|
| <i>base</i>               | LPSPI peripheral address.                                                                   |
| <i>delayTimeInNanoSec</i> | The desired delay value in nano-seconds.                                                    |
| <i>whichDelay</i>         | The desired delay to configuration, which must be of type <code>lpspi_delay_type_t</code> . |
| <i>srcClock_Hz</i>        | Module source input clock in Hertz.                                                         |

Returns

actual Calculated delay value in nano-seconds.

**31.2.8.33** `static void LPSPI_WriteData ( LPSPI_Type * base, uint32_t data ) [inline], [static]`

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | LPSPI peripheral address. |
| <i>data</i> | The data word to be sent. |

**31.2.8.34** `static uint32_t LPSPI_ReadData ( LPSPI_Type * base ) [inline], [static]`

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

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

Returns

The data read from the data buffer.

**31.2.8.35** `void LPSPI_SetDummyData ( LPSPI_Type * base, uint8_t dummyData )`

## Parameters

|                  |                                                                                                                                                                                                                                                 |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | LPSPI peripheral address.                                                                                                                                                                                                                       |
| <i>dummyData</i> | Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated. |

**31.2.8.36 void LPSPI\_MasterTransferCreateHandle ( LPSPI\_Type \* *base*,  
lpspi\_master\_handle\_t \* *handle*, lpspi\_master\_transfer\_callback\_t *callback*,  
void \* *userData* )**

This function initializes the LPSPI handle, which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

## Parameters

|                 |                                                |
|-----------------|------------------------------------------------|
| <i>base</i>     | LPSPI peripheral address.                      |
| <i>handle</i>   | LPSPI handle pointer to lpspi_master_handle_t. |
| <i>callback</i> | DSPI callback.                                 |
| <i>userData</i> | callback function parameter.                   |

**31.2.8.37 status\_t LPSPI\_MasterTransferBlocking ( LPSPI\_Type \* *base*, lpspi\_transfer\_t  
\* *transfer* )**

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

## Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral address.                              |
| <i>transfer</i> | pointer to <a href="#">lpspi_transfer_t</a> structure. |

## Returns

status of status\_t.

### 31.2.8.38 `status_t LPSPI_MasterTransferNonBlocking ( LPSPI_Type * base, lpspi_master_handle_t * handle, lpspi_transfer_t * transfer )`

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

|                 |                                                                                          |
|-----------------|------------------------------------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral address.                                                                |
| <i>handle</i>   | pointer to <code>lpspi_master_handle_t</code> structure which stores the transfer state. |
| <i>transfer</i> | pointer to <code>lpspi_transfer_t</code> structure.                                      |

Returns

status of `status_t`.

### 31.2.8.39 `status_t LPSPI_MasterTransferGetCount ( LPSPI_Type * base, lpspi_master_handle_t * handle, size_t * count )`

This function gets the master transfer remaining bytes.

Parameters

|               |                                                                                          |
|---------------|------------------------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral address.                                                                |
| <i>handle</i> | pointer to <code>lpspi_master_handle_t</code> structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.                      |

Returns

status of `status_t`.

### 31.2.8.40 `void LPSPI_MasterTransferAbort ( LPSPI_Type * base, lpspi_master_handle_t * handle )`

This function aborts a transfer which uses an interrupt method.

Parameters

|               |                                                                                          |
|---------------|------------------------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral address.                                                                |
| <i>handle</i> | pointer to <code>lpspi_master_handle_t</code> structure which stores the transfer state. |

#### 31.2.8.41 void LPSPI\_MasterTransferHandleIRQ ( LPSPI\_Type \* *base*, lpspi\_master\_handle\_t \* *handle* )

This function processes the LPSPI transmit and receive IRQ.

Parameters

|               |                                                                                          |
|---------------|------------------------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral address.                                                                |
| <i>handle</i> | pointer to <code>lpspi_master_handle_t</code> structure which stores the transfer state. |

#### 31.2.8.42 void LPSPI\_SlaveTransferCreateHandle ( LPSPI\_Type \* *base*, lpspi\_slave\_handle\_t \* *handle*, lpspi\_slave\_transfer\_callback\_t *callback*, void \* *userData* )

This function initializes the LPSPI handle, which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Parameters

|                 |                                                             |
|-----------------|-------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral address.                                   |
| <i>handle</i>   | LPSPI handle pointer to <code>lpspi_slave_handle_t</code> . |
| <i>callback</i> | DSPI callback.                                              |
| <i>userData</i> | callback function parameter.                                |

#### 31.2.8.43 status\_t LPSPI\_SlaveTransferNonBlocking ( LPSPI\_Type \* *base*, lpspi\_slave\_handle\_t \* *handle*, lpspi\_transfer\_t \* *transfer* )

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

|                 |                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral address.                                                               |
| <i>handle</i>   | pointer to <code>lpspi_slave_handle_t</code> structure which stores the transfer state. |
| <i>transfer</i> | pointer to <code>lpspi_transfer_t</code> structure.                                     |

Returns

status of `status_t`.

**31.2.8.44** `status_t LPSPI_SlaveTransferGetCount ( LPSPI_Type * base, lpspi_slave_handle_t * handle, size_t * count )`

This function gets the slave transfer remaining bytes.

Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral address.                                                               |
| <i>handle</i> | pointer to <code>lpspi_slave_handle_t</code> structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.                     |

Returns

status of `status_t`.

**31.2.8.45** `void LPSPI_SlaveTransferAbort ( LPSPI_Type * base, lpspi_slave_handle_t * handle )`

This function aborts a transfer which uses an interrupt method.

Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral address.                                                               |
| <i>handle</i> | pointer to <code>lpspi_slave_handle_t</code> structure which stores the transfer state. |

**31.2.8.46** `void LPSPI_SlaveTransferHandleIRQ ( LPSPI_Type * base, lpspi_slave_handle_t * handle )`

This function processes the LPSPI transmit and receives an IRQ.



Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral address.                                                               |
| <i>handle</i> | pointer to <code>lpspi_slave_handle_t</code> structure which stores the transfer state. |

## 31.2.9 Variable Documentation

### 31.2.9.1 `volatile uint8_t g_lpspiDummyData[]`

## 31.3 LPSPI eDMA Driver

### 31.3.1 Overview

#### Data Structures

- struct `lpspi_master_edma_handle_t`  
*LPSPI master eDMA transfer handle structure used for transactional API. [More...](#)*
- struct `lpspi_slave_edma_handle_t`  
*LPSPI slave eDMA transfer handle structure used for transactional API. [More...](#)*

#### Typedefs

- typedef void(\* `lpspi_master_edma_transfer_callback_t`)(LPSPI\_Type \*base, lpspi\_master\_edma\_handle\_t \*handle, `status_t` status, void \*userData)  
*Completion callback function pointer type.*
- typedef void(\* `lpspi_slave_edma_transfer_callback_t`)(LPSPI\_Type \*base, lpspi\_slave\_edma\_handle\_t \*handle, `status_t` status, void \*userData)  
*Completion callback function pointer type.*

#### Functions

- void `LPSPI_MasterTransferCreateHandleEDMA` (LPSPI\_Type \*base, lpspi\_master\_edma\_handle\_t \*handle, `lpspi_master_edma_transfer_callback_t` callback, void \*userData, `edma_handle_t` \*edmaRxRegToRxDataHandle, `edma_handle_t` \*edmaTxDataToTxRegHandle)  
*Initializes the LPSPI master eDMA handle.*
- `status_t` `LPSPI_MasterTransferEDMA` (LPSPI\_Type \*base, lpspi\_master\_edma\_handle\_t \*handle, `lpspi_transfer_t` \*transfer)  
*LPSPI master transfer data using eDMA.*
- `status_t` `LPSPI_MasterTransferPrepareEDMALite` (LPSPI\_Type \*base, lpspi\_master\_edma\_handle\_t \*handle, uint32\_t configFlags)  
*LPSPI master config transfer parameter while using eDMA.*
- `status_t` `LPSPI_MasterTransferEDMALite` (LPSPI\_Type \*base, lpspi\_master\_edma\_handle\_t \*handle, `lpspi_transfer_t` \*transfer)  
*LPSPI master transfer data using eDMA without configs.*
- void `LPSPI_MasterTransferAbortEDMA` (LPSPI\_Type \*base, lpspi\_master\_edma\_handle\_t \*handle)  
*LPSPI master aborts a transfer which is using eDMA.*
- `status_t` `LPSPI_MasterTransferGetCountEDMA` (LPSPI\_Type \*base, lpspi\_master\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets the master eDMA transfer remaining bytes.*
- void `LPSPI_SlaveTransferCreateHandleEDMA` (LPSPI\_Type \*base, lpspi\_slave\_edma\_handle\_t \*handle, `lpspi_slave_edma_transfer_callback_t` callback, void \*userData, `edma_handle_t` \*edmaRxRegToRxDataHandle, `edma_handle_t` \*edmaTxDataToTxRegHandle)  
*Initializes the LPSPI slave eDMA handle.*

- `status_t LPSPI_SlaveTransferEDMA` (LPSPI\_Type \*base, lpspi\_slave\_edma\_handle\_t \*handle, lpspi\_transfer\_t \*transfer)  
*LPSPI slave transfers data using eDMA.*
- `void LPSPI_SlaveTransferAbortEDMA` (LPSPI\_Type \*base, lpspi\_slave\_edma\_handle\_t \*handle)  
*LPSPI slave aborts a transfer which is using eDMA.*
- `status_t LPSPI_SlaveTransferGetCountEDMA` (LPSPI\_Type \*base, lpspi\_slave\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets the slave eDMA transfer remaining bytes.*

## Driver version

- `#define FSL_LPSPI_EDMA_DRIVER_VERSION` (MAKE\_VERSION(2, 4, 0))  
*LPSPI EDMA driver version.*

## 31.3.2 Data Structure Documentation

### 31.3.2.1 struct\_lpspi\_master\_edma\_handle

Forward declaration of the `_lpspi_master_edma_handle` typedefs.

#### Data Fields

- volatile bool `isPcsContinuous`  
*Is PCS continuous in transfer.*
- volatile bool `isByteSwap`  
*A flag that whether should byte swap.*
- volatile uint8\_t `fifoSize`  
*FIFO dataSize.*
- volatile uint8\_t `rxWatermark`  
*Rx watermark.*
- volatile uint8\_t `bytesEachWrite`  
*Bytes for each write TDR.*
- volatile uint8\_t `bytesEachRead`  
*Bytes for each read RDR.*
- volatile uint8\_t `bytesLastRead`  
*Bytes for last read RDR.*
- volatile bool `isThereExtraRxBytes`  
*Is there extra RX byte.*
- uint8\_t \*volatile `txData`  
*Send buffer.*
- uint8\_t \*volatile `rxData`  
*Receive buffer.*
- volatile size\_t `txRemainingByteCount`  
*Number of bytes remaining to send.*
- volatile size\_t `rxRemainingByteCount`  
*Number of bytes remaining to receive.*
- volatile uint32\_t `writeRegRemainingTimes`

- Write TDR register remaining times.*

  - volatile uint32\_t [readRegRemainingTimes](#)

*Read RDR register remaining times.*
- uint32\_t [totalByteCount](#)
- Number of transfer bytes.*

  - uint32\_t [txBuffIfNull](#)

*Used if there is not txData for DMA purpose.*
- uint32\_t [rxBuffIfNull](#)
- Used if there is not rxData for DMA purpose.*

  - uint32\_t [transmitCommand](#)

*Used to write TCR for DMA purpose.*
- volatile uint8\_t [state](#)
- LPSPI transfer state , \_lpspi\_transfer\_state.*

  - uint8\_t [nbytes](#)

*eDMA minor byte transfer count initially configured.*
- [lpspi\\_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](#) \* [edmaTxDataToTxRegHandle](#)

*edma\_handle\_t handle point used for TxData to TxReg buff*
- [edma\\_tcd\\_t](#) [lpspiSoftwareTCD](#) [3]
- SoftwareTCD, internal used.*

## Field Documentation

- (1) `volatile bool lpspi_master_edma_handle_t::isPcsContinuous`
- (2) `volatile bool lpspi_master_edma_handle_t::isByteSwap`
- (3) `volatile uint8_t lpspi_master_edma_handle_t::fifoSize`
- (4) `volatile uint8_t lpspi_master_edma_handle_t::rxWatermark`
- (5) `volatile uint8_t lpspi_master_edma_handle_t::bytesEachWrite`
- (6) `volatile uint8_t lpspi_master_edma_handle_t::bytesEachRead`
- (7) `volatile uint8_t lpspi_master_edma_handle_t::bytesLastRead`
- (8) `volatile bool lpspi_master_edma_handle_t::isThereExtraRxBytes`
- (9) `uint8_t* volatile lpspi_master_edma_handle_t::txData`
- (10) `uint8_t* volatile lpspi_master_edma_handle_t::rxData`
- (11) `volatile size_t lpspi_master_edma_handle_t::txRemainingByteCount`
- (12) `volatile size_t lpspi_master_edma_handle_t::rxRemainingByteCount`
- (13) `volatile uint32_t lpspi_master_edma_handle_t::writeRegRemainingTimes`
- (14) `volatile uint32_t lpspi_master_edma_handle_t::readRegRemainingTimes`
- (15) `uint32_t lpspi_master_edma_handle_t::txBuffIfNull`
- (16) `uint32_t lpspi_master_edma_handle_t::rxBuffIfNull`
- (17) `uint32_t lpspi_master_edma_handle_t::transmitCommand`
- (18) `volatile uint8_t lpspi_master_edma_handle_t::state`
- (19) `uint8_t lpspi_master_edma_handle_t::nbytes`
- (20) `lpspi_master_edma_transfer_callback_t lpspi_master_edma_handle_t::callback`
- (21) `void* lpspi_master_edma_handle_t::userData`

### 31.3.2.2 struct `_lpspi_slave_edma_handle`

Forward declaration of the [\\_lpspi\\_slave\\_edma\\_handle](#) typedefs.

## Data Fields

- volatile bool `isByteSwap`  
*A flag that whether should byte swap.*
- volatile uint8\_t `fifoSize`  
*FIFO dataSize.*
- volatile uint8\_t `rxWatermark`  
*Rx watermark.*
- volatile uint8\_t `bytesEachWrite`  
*Bytes for each write TDR.*
- volatile uint8\_t `bytesEachRead`  
*Bytes for each read RDR.*
- volatile uint8\_t `bytesLastRead`  
*Bytes for last read RDR.*
- volatile bool `isThereExtraRxBytes`  
*Is there extra RX byte.*
- uint8\_t `nbytes`  
*eDMA minor byte transfer count initially configured.*
- uint8\_t \*volatile `txData`  
*Send buffer.*
- uint8\_t \*volatile `rxData`  
*Receive buffer.*
- volatile size\_t `txRemainingByteCount`  
*Number of bytes remaining to send.*
- volatile size\_t `rxRemainingByteCount`  
*Number of bytes remaining to receive.*
- volatile uint32\_t `writeRegRemainingTimes`  
*Write TDR register remaining times.*
- volatile uint32\_t `readRegRemainingTimes`  
*Read RDR register remaining times.*
- uint32\_t `totalByteCount`  
*Number of transfer bytes.*
- uint32\_t `txBuffIfNull`  
*Used if there is not txData for DMA purpose.*
- uint32\_t `rxBuffIfNull`  
*Used if there is not rxData for DMA purpose.*
- volatile uint8\_t `state`  
*LPSPI transfer state.*
- uint32\_t `errorCount`  
*Error count for slave transfer.*
- `lpspi_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*
- `edma_tcd_t` `lpspiSoftwareTCD` [2]  
*SoftwareTCD, internal used.*



**Field Documentation**

- (1) `volatile bool lpspi_slave_edma_handle_t::isByteSwap`
- (2) `volatile uint8_t lpspi_slave_edma_handle_t::fifoSize`
- (3) `volatile uint8_t lpspi_slave_edma_handle_t::rxWatermark`
- (4) `volatile uint8_t lpspi_slave_edma_handle_t::bytesEachWrite`
- (5) `volatile uint8_t lpspi_slave_edma_handle_t::bytesEachRead`
- (6) `volatile uint8_t lpspi_slave_edma_handle_t::bytesLastRead`
- (7) `volatile bool lpspi_slave_edma_handle_t::isThereExtraRxBytes`
- (8) `uint8_t lpspi_slave_edma_handle_t::nbytes`
- (9) `uint8_t* volatile lpspi_slave_edma_handle_t::txData`
- (10) `uint8_t* volatile lpspi_slave_edma_handle_t::rxData`
- (11) `volatile size_t lpspi_slave_edma_handle_t::txRemainingByteCount`
- (12) `volatile size_t lpspi_slave_edma_handle_t::rxRemainingByteCount`
- (13) `volatile uint32_t lpspi_slave_edma_handle_t::writeRegRemainingTimes`
- (14) `volatile uint32_t lpspi_slave_edma_handle_t::readRegRemainingTimes`
- (15) `uint32_t lpspi_slave_edma_handle_t::txBuffIfNull`
- (16) `uint32_t lpspi_slave_edma_handle_t::rxBuffIfNull`
- (17) `volatile uint8_t lpspi_slave_edma_handle_t::state`
- (18) `uint32_t lpspi_slave_edma_handle_t::errorCount`
- (19) `lpspi_slave_edma_transfer_callback_t lpspi_slave_edma_handle_t::callback`
- (20) `void* lpspi_slave_edma_handle_t::userData`

**31.3.3 Macro Definition Documentation**

**31.3.3.1** `#define FSL_LPSPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))`

**31.3.4 Typedef Documentation**

**31.3.4.1** `typedef void(* lpspi_master_edma_transfer_callback_t)(LPSPI_Type *base, lpspi_master_edma_handle_t *handle, status_t status, void *userData)`



## Parameters

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

### 31.3.4.2 typedef void(\* lpspi\_slave\_edma\_transfer\_callback\_t)(LPSPI\_Type \*base, lpspi\_slave\_edma\_handle\_t \*handle, status\_t status, void \*userData)

## Parameters

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

## 31.3.5 Function Documentation

### 31.3.5.1 void LPSPI\_MasterTransferCreateHandleEDMA ( LPSPI\_Type \* base, lpspi\_master\_edma\_handle\_t \* handle, lpspi\_master\_edma\_transfer\_callback\_t callback, void \* userData, edma\_handle\_t \* edmaRxRegToRxDataHandle, edma\_handle\_t \* edmaTxDataToTxRegHandle )

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that the LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Tx DMAMUX source for edmaRxRegToRxDataHandle.

## Parameters

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

|                                  |                                                                                 |
|----------------------------------|---------------------------------------------------------------------------------|
| <i>handle</i>                    | LPSPI handle pointer to <code>lpspi_master_edma_handle_t</code> .               |
| <i>callback</i>                  | LPSPI callback.                                                                 |
| <i>userData</i>                  | callback function parameter.                                                    |
| <i>edmaRxRegTo-RxDataHandle</i>  | <code>edmaRxRegToRxDataHandle</code> pointer to <a href="#">edma_handle_t</a> . |
| <i>edmaTxData-ToTxReg-Handle</i> | <code>edmaTxDataToTxRegHandle</code> pointer to <a href="#">edma_handle_t</a> . |

### 31.3.5.2 `status_t LPSPI_MasterTransferEDMA ( LPSPI_Type * base, lpspi_master_edma_handle_t * handle, lpspi_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 transfer data size should be an integer multiple of `bytesPerFrame` if `bytesPerFrame` is less than or equal to 4. For `bytesPerFrame` greater than 4: The transfer data size should be equal to `bytesPerFrame` if the `bytesPerFrame` is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of `bytesPerFrame`.

Parameters

|                 |                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral base address.                                                                |
| <i>handle</i>   | pointer to <code>lpspi_master_edma_handle_t</code> structure which stores the transfer state. |
| <i>transfer</i> | pointer to <a href="#">lpspi_transfer_t</a> structure.                                        |

Returns

status of `status_t`.

### 31.3.5.3 `status_t LPSPI_MasterTransferPrepareEDMALite ( LPSPI_Type * base, lpspi_master_edma_handle_t * handle, uint32_t configFlags )`

This function is preparing to transfer data using eDMA, work with `LPSPI_MasterTransferEDMALite`.

Parameters

|                    |                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>        | LPSPI peripheral base address.                                                                |
| <i>handle</i>      | pointer to <code>lpspi_master_edma_handle_t</code> structure which stores the transfer state. |
| <i>configFlags</i> | transfer configuration flags. <code>_lpspi_transfer_config_flag_for_master</code> .           |

## Returns

Indicates whether LPSPI master transfer was successful or not.

## Return values

|                           |                           |
|---------------------------|---------------------------|
| <i>kStatus_Success</i>    | Execution successfully.   |
| <i>kStatus_LPSPI_Busy</i> | The LPSPI device is busy. |

#### 31.3.5.4 `status_t LPSPI_MasterTransferEDMALite ( LPSPI_Type * base, lpspi_master_edma_handle_t * handle, lpspi_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: This API is only for transfer through DMA without configuration. Before calling this API, you must call `LPSPI_MasterTransferPrepareEDMALite` to configure it once. The transfer data size should be an integer multiple of `bytesPerFrame` if `bytesPerFrame` is less than or equal to 4. For `bytesPerFrame` greater than 4: The transfer data size should be equal to `bytesPerFrame` if the `bytesPerFrame` is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of `bytesPerFrame`.

## Parameters

|                 |                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral base address.                                                                |
| <i>handle</i>   | pointer to <code>lpspi_master_edma_handle_t</code> structure which stores the transfer state. |
| <i>transfer</i> | pointer to <code>lpspi_transfer_t</code> structure, <code>config</code> field is not used.    |

## Returns

Indicates whether LPSPI master transfer was successful or not.

## Return values

|                                |                                    |
|--------------------------------|------------------------------------|
| <i>kStatus_Success</i>         | Execution successfully.            |
| <i>kStatus_LPSPI_Busy</i>      | The LPSPI device is busy.          |
| <i>kStatus_InvalidArgument</i> | The transfer structure is invalid. |

### 31.3.5.5 void LPSPI\_MasterTransferAbortEDMA ( LPSPI\_Type \* *base*, lpspi\_master\_edma\_handle\_t \* *handle* )

This function aborts a transfer which is using eDMA.

Parameters

|               |                                                                                  |
|---------------|----------------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral base address.                                                   |
| <i>handle</i> | pointer to lpspi_master_edma_handle_t structure which stores the transfer state. |

### 31.3.5.6 status\_t LPSPI\_MasterTransferGetCountEDMA ( LPSPI\_Type \* *base*, lpspi\_master\_edma\_handle\_t \* *handle*, size\_t \* *count* )

This function gets the master eDMA transfer remaining bytes.

Parameters

|               |                                                                                  |
|---------------|----------------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral base address.                                                   |
| <i>handle</i> | pointer to lpspi_master_edma_handle_t structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the EDMA transaction.                      |

Returns

status of status\_t.

### 31.3.5.7 void LPSPI\_SlaveTransferCreateHandleEDMA ( LPSPI\_Type \* *base*, lpspi\_slave\_edma\_handle\_t \* *handle*, lpspi\_slave\_edma\_transfer\_callback\_t *callback*, void \* *userData*, edma\_handle\_t \* *edmaRxRegToRxDataHandle*, edma\_handle\_t \* *edmaTxDataToTxRegHandle* )

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRx-DataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for edmaRxRegToRxDataHandle .

## Parameters

|                                  |                                                                                 |
|----------------------------------|---------------------------------------------------------------------------------|
| <i>base</i>                      | LPSPI peripheral base address.                                                  |
| <i>handle</i>                    | LPSPI handle pointer to <code>lpspi_slave_edma_handle_t</code> .                |
| <i>callback</i>                  | LPSPI callback.                                                                 |
| <i>userData</i>                  | callback function parameter.                                                    |
| <i>edmaRxRegTo-RxDataHandle</i>  | <code>edmaRxRegToRxDataHandle</code> pointer to <a href="#">edma_handle_t</a> . |
| <i>edmaTxData-ToTxReg-Handle</i> | <code>edmaTxDataToTxRegHandle</code> pointer to <a href="#">edma_handle_t</a> . |

### 31.3.5.8 `status_t LPSPI_SlaveTransferEDMA ( LPSPI_Type * base, lpspi_slave_edma_handle_t * handle, lpspi_transfer_t * transfer )`

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

## Parameters

|                 |                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral base address.                                                               |
| <i>handle</i>   | pointer to <code>lpspi_slave_edma_handle_t</code> structure which stores the transfer state. |
| <i>transfer</i> | pointer to <a href="#">lpspi_transfer_t</a> structure.                                       |

## Returns

status of `status_t`.

### 31.3.5.9 `void LPSPI_SlaveTransferAbortEDMA ( LPSPI_Type * base, lpspi_slave_edma_handle_t * handle )`

This function aborts a transfer which is using eDMA.

Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral base address.                                                               |
| <i>handle</i> | pointer to <code>lpspi_slave_edma_handle_t</code> structure which stores the transfer state. |

### 31.3.5.10 `status_t LPSPI_SlaveTransferGetCountEDMA ( LPSPI_Type * base, lpspi_slave_edma_handle_t * handle, size_t * count )`

This function gets the slave eDMA transfer remaining bytes.

Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral base address.                                                               |
| <i>handle</i> | pointer to <code>lpspi_slave_edma_handle_t</code> structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the eDMA transaction.                                  |

Returns

status of `status_t`.

## 31.4 LPSPI FreeRTOS Driver

### 31.4.1 Overview

#### Driver version

- #define `FSL_LPSPI_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 1)`)  
*LPSPI FreeRTOS driver version 2.3.1.*

#### LPSPI RTOS Operation

- `status_t LPSPI_RTOS_Init` (`lpspi_rtos_handle_t *handle`, `LPSPI_Type *base`, `const lpspi_master_config_t *masterConfig`, `uint32_t srcClock_Hz`)  
*Initializes LPSPI.*
- `status_t LPSPI_RTOS_Deinit` (`lpspi_rtos_handle_t *handle`)  
*Deinitializes the LPSPI.*
- `status_t LPSPI_RTOS_Transfer` (`lpspi_rtos_handle_t *handle`, `lpspi_transfer_t *transfer`)  
*Performs SPI transfer.*

### 31.4.2 Macro Definition Documentation

#### 31.4.2.1 #define FSL\_LPSPI\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 1))

### 31.4.3 Function Documentation

#### 31.4.3.1 `status_t LPSPI_RTOS_Init ( lpspi_rtos_handle_t * handle, LPSPI_Type * base, const lpspi_master_config_t * masterConfig, uint32_t srcClock_Hz )`

This function initializes the LPSPI module and related RTOS context.

Parameters

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

Returns

status of the operation.



**31.4.3.2 status\_t LPSPI\_RTOS\_Deinit ( lpspi\_rtos\_handle\_t \* *handle* )**

This function deinitializes the LPSPI module and related RTOS context.

## Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | The RTOS LPSPI handle. |
|---------------|------------------------|

### 31.4.3.3 `status_t LPSPI_RTOS_Transfer ( lpspi_rtos_handle_t * handle, lpspi_transfer_t * transfer )`

This function performs an SPI transfer according to data given in the transfer structure.

## Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>handle</i>   | The RTOS LPSPI handle.                        |
| <i>transfer</i> | Structure specifying the transfer parameters. |

## Returns

status of the operation.

## 31.5 LPSPI CMSIS Driver

This section describes the programming interface of the LPSPI Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method please refer to <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

### 31.5.1 Function groups

#### 31.5.1.1 LPSPI CMSIS GetVersion Operation

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

#### 31.5.1.2 LPSPI CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

#### 31.5.1.3 LPSPI 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.

#### 31.5.1.4 LPSPI Transfer Operation

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

#### 31.5.1.5 LPSPI Status Operation

This function group gets the LPSPI transfer status.

#### 31.5.1.6 LPSPI CMSIS Control Operation

This function can select 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 set other control command.

## 31.5.2 Typical use case

### 31.5.2.1 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();

```

### 31.5.2.2 Slave Operation

```

/* Variables */
uint8_t slaveRxData[TRANSFER_SIZE] = {0U};
uint8_t slaveTxData[TRANSFER_SIZE] = {0U};

/*DSPI slave init*/
Driver_SPI2.Initialize(DSPI_SlaveSignalEvent_t);
Driver_SPI2.PowerControl(ARM_POWER_FULL);
Driver_SPI2.Control(ARM_SPI_MODE_SLAVE, false);

/* Start slave transfer */
Driver_SPI2.Transfer(slaveTxData, slaveRxData, TRANSFER_SIZE);

/* slave power off */
Driver_SPI2.PowerControl(ARM_POWER_OFF);

/* slave uninitialized */
Driver_SPI2.Uninitialize();

```



## Chapter 32

# LPUART: Low Power Universal Asynchronous Receiver/- Transmitter Driver

### 32.1 Overview

#### Modules

- [LPUART CMSIS Driver](#)
- [LPUART Driver](#)
- [LPUART FreeRTOS Driver](#)
- [LPUART eDMA Driver](#)

## 32.2 LPUART Driver

### 32.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power UART (LPUART) module of MCUXpresso SDK devices.

### 32.2.2 Typical use case

#### 32.2.2.1 LPUART Operation

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

### Data Structures

- struct [lpuart\\_config\\_t](#)  
*LPUART configuration structure. [More...](#)*
- struct [lpuart\\_transfer\\_t](#)  
*LPUART transfer structure. [More...](#)*
- struct [lpuart\\_handle\\_t](#)  
*LPUART handle structure. [More...](#)*

### Macros

- #define [UART\\_RETRY\\_TIMES](#) 0U /\* Defining to zero means to keep waiting for the flag until it is assert/deassert. \*/  
*Retry times for waiting flag.*

### Typedefs

- typedef void(\* [lpuart\\_transfer\\_callback\\_t](#))(LPUART\_Type \*base, lpuart\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*LPUART transfer callback function.*

## Enumerations

- enum {
  - kStatus\_LPUART\_TxBusy = MAKE\_STATUS(kStatusGroup\_LPUART, 0),
  - kStatus\_LPUART\_RxBusy = MAKE\_STATUS(kStatusGroup\_LPUART, 1),
  - kStatus\_LPUART\_TxIdle = MAKE\_STATUS(kStatusGroup\_LPUART, 2),
  - kStatus\_LPUART\_RxIdle = MAKE\_STATUS(kStatusGroup\_LPUART, 3),
  - kStatus\_LPUART\_TxWatermarkTooLarge = MAKE\_STATUS(kStatusGroup\_LPUART, 4),
  - kStatus\_LPUART\_RxWatermarkTooLarge = MAKE\_STATUS(kStatusGroup\_LPUART, 5),
  - kStatus\_LPUART\_FlagCannotClearManually = MAKE\_STATUS(kStatusGroup\_LPUART, 6),
  - kStatus\_LPUART\_Error = MAKE\_STATUS(kStatusGroup\_LPUART, 7),
  - kStatus\_LPUART\_RxRingBufferOverrun,
  - kStatus\_LPUART\_RxHardwareOverrun = MAKE\_STATUS(kStatusGroup\_LPUART, 9),
  - kStatus\_LPUART\_NoiseError = MAKE\_STATUS(kStatusGroup\_LPUART, 10),
  - kStatus\_LPUART\_FramingError = MAKE\_STATUS(kStatusGroup\_LPUART, 11),
  - kStatus\_LPUART\_ParityError = MAKE\_STATUS(kStatusGroup\_LPUART, 12),
  - kStatus\_LPUART\_BaudrateNotSupport,
  - kStatus\_LPUART\_IdleLineDetected = MAKE\_STATUS(kStatusGroup\_LPUART, 14),
  - kStatus\_LPUART\_Timeout = MAKE\_STATUS(kStatusGroup\_LPUART, 15) }

*Error codes for the LPUART driver.*
- enum lpuart\_parity\_mode\_t {
  - kLPUART\_ParityDisabled = 0x0U,
  - kLPUART\_ParityEven = 0x2U,
  - kLPUART\_ParityOdd = 0x3U }

*LPUART parity mode.*
- enum lpuart\_data\_bits\_t { kLPUART\_EightDataBits = 0x0U }

*LPUART data bits count.*

- enum lpuart\_stop\_bit\_count\_t {
  - kLPUART\_OneStopBit = 0U,
  - kLPUART\_TwoStopBit = 1U }

*LPUART stop bit count.*
- enum lpuart\_idle\_type\_select\_t {
  - kLPUART\_IdleTypeStartBit = 0U,
  - kLPUART\_IdleTypeStopBit = 1U }

*LPUART idle flag type defines when the receiver starts counting.*
- enum lpuart\_idle\_config\_t {
  - kLPUART\_IdleCharacter1 = 0U,
  - kLPUART\_IdleCharacter2 = 1U,
  - kLPUART\_IdleCharacter4 = 2U,
  - kLPUART\_IdleCharacter8 = 3U,
  - kLPUART\_IdleCharacter16 = 4U,
  - kLPUART\_IdleCharacter32 = 5U,
  - kLPUART\_IdleCharacter64 = 6U,
  - kLPUART\_IdleCharacter128 = 7U }

*LPUART idle detected configuration.*
- enum \_lpuart\_interrupt\_enable {

```

kLPUART_RxActiveEdgeInterruptEnable = (LPUART_BAUD_RXEDGIE_MASK >> 8U),
kLPUART_TxDataRegEmptyInterruptEnable = (LPUART_CTRL_TIE_MASK),
kLPUART_TransmissionCompleteInterruptEnable = (LPUART_CTRL_TCIE_MASK),
kLPUART_RxDataRegFullInterruptEnable = (LPUART_CTRL_RIE_MASK),
kLPUART_IdleLineInterruptEnable = (LPUART_CTRL_ILIE_MASK),
kLPUART_RxOverrunInterruptEnable = (LPUART_CTRL_ORIE_MASK),
kLPUART_NoiseErrorInterruptEnable = (LPUART_CTRL_NEIE_MASK),
kLPUART_FramingErrorInterruptEnable = (LPUART_CTRL_FEIE_MASK),
kLPUART_ParityErrorInterruptEnable = (LPUART_CTRL_PEIE_MASK) }

```

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

- enum `_lpuart_flags` {
 

```

kLPUART_TxDataRegEmptyFlag,
kLPUART_TransmissionCompleteFlag,
kLPUART_RxDataRegFullFlag = (LPUART_STAT_RDRF_MASK),
kLPUART_IdleLineFlag = (LPUART_STAT_IDLE_MASK),
kLPUART_RxOverrunFlag = (LPUART_STAT_OR_MASK),
kLPUART_NoiseErrorFlag = (LPUART_STAT_NF_MASK),
kLPUART_FramingErrorFlag,
kLPUART_ParityErrorFlag = (LPUART_STAT_PF_MASK),
kLPUART_RxActiveEdgeFlag = (LPUART_STAT_RXEDGIF_MASK),
kLPUART_RxActiveFlag }

```

*LPUART status flags.*

## Driver version

- #define `FSL_LPUART_DRIVER_VERSION` (`MAKE_VERSION(2, 7, 3)`)  
*LPUART driver version.*

## Initialization and deinitialization

- `status_t LPUART_Init` (`LPUART_Type *base`, `const lpuart_config_t *config`, `uint32_t srcClock_Hz`)  
*Initializes an LPUART instance with the user configuration structure and the peripheral clock.*
- `void LPUART_Deinit` (`LPUART_Type *base`)  
*Deinitializes a LPUART instance.*
- `void LPUART_GetDefaultConfig` (`lpuart_config_t *config`)  
*Gets the default configuration structure.*

## Module configuration

- `status_t LPUART_SetBaudRate` (`LPUART_Type *base`, `uint32_t baudRate_Bps`, `uint32_t srcClock_Hz`)  
*Sets the LPUART instance baudrate.*
- `void LPUART_Enable9bitMode` (`LPUART_Type *base`, `bool enable`)



- *Enable 9-bit data mode for LPUART.*
- static void [LPUART\\_SetMatchAddress](#) (LPUART\_Type \*base, uint16\_t address1, uint16\_t address2)  
*Set the LPUART address.*
- static void [LPUART\\_EnableMatchAddress](#) (LPUART\_Type \*base, bool match1, bool match2)  
*Enable the LPUART match address feature.*

## Status

- uint32\_t [LPUART\\_GetStatusFlags](#) (LPUART\_Type \*base)  
*Gets LPUART status flags.*
- [status\\_t LPUART\\_ClearStatusFlags](#) (LPUART\_Type \*base, uint32\_t mask)  
*Clears status flags with a provided mask.*

## Interrupts

- void [LPUART\\_EnableInterrupts](#) (LPUART\_Type \*base, uint32\_t mask)  
*Enables LPUART interrupts according to a provided mask.*
- void [LPUART\\_DisableInterrupts](#) (LPUART\_Type \*base, uint32\_t mask)  
*Disables LPUART interrupts according to a provided mask.*
- uint32\_t [LPUART\\_GetEnabledInterrupts](#) (LPUART\_Type \*base)  
*Gets enabled LPUART interrupts.*

## Bus Operations

- uint32\_t [LPUART\\_GetInstance](#) (LPUART\_Type \*base)  
*Get the LPUART instance from peripheral base address.*
- static void [LPUART\\_EnableTx](#) (LPUART\_Type \*base, bool enable)  
*Enables or disables the LPUART transmitter.*
- static void [LPUART\\_EnableRx](#) (LPUART\_Type \*base, bool enable)  
*Enables or disables the LPUART receiver.*
- static void [LPUART\\_WriteByte](#) (LPUART\_Type \*base, uint8\_t data)  
*Writes to the transmitter register.*
- static uint8\_t [LPUART\\_ReadByte](#) (LPUART\_Type \*base)  
*Reads the receiver register.*
- void [LPUART\\_SendAddress](#) (LPUART\_Type \*base, uint8\_t address)  
*Transmit an address frame in 9-bit data mode.*
- [status\\_t LPUART\\_WriteBlocking](#) (LPUART\_Type \*base, const uint8\_t \*data, size\_t length)  
*Writes to the transmitter register using a blocking method.*
- [status\\_t LPUART\\_ReadBlocking](#) (LPUART\_Type \*base, uint8\_t \*data, size\_t length)  
*Reads the receiver data register using a blocking method.*

## Transactional

- void [LPUART\\_TransferCreateHandle](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle, [lpuart\\_transfer\\_callback\\_t](#) callback, void \*userData)

- Initializes the LPUART handle.*

  - `status_t LPUART_TransferSendNonBlocking` (LPUART\_Type \*base, lpuart\_handle\_t \*handle, lpuart\_transfer\_t \*xfer)

*Transmits a buffer of data using the interrupt method.*
- `void LPUART_TransferStartRingBuffer` (LPUART\_Type \*base, lpuart\_handle\_t \*handle, uint8\_t \*ringBuffer, size\_t ringBufferSize)

*Sets up the RX ring buffer.*
- `void LPUART_TransferStopRingBuffer` (LPUART\_Type \*base, lpuart\_handle\_t \*handle)

*Aborts the background transfer and uninstalls the ring buffer.*
- `size_t LPUART_TransferGetRxRingBufferLength` (LPUART\_Type \*base, lpuart\_handle\_t \*handle)

*Get the length of received data in RX ring buffer.*
- `void LPUART_TransferAbortSend` (LPUART\_Type \*base, lpuart\_handle\_t \*handle)

*Aborts the interrupt-driven data transmit.*
- `status_t LPUART_TransferGetSendCount` (LPUART\_Type \*base, lpuart\_handle\_t \*handle, uint32\_t \*count)

*Gets the number of bytes that have been sent out to bus.*
- `status_t LPUART_TransferReceiveNonBlocking` (LPUART\_Type \*base, lpuart\_handle\_t \*handle, lpuart\_transfer\_t \*xfer, size\_t \*receivedBytes)

*Receives a buffer of data using the interrupt method.*
- `void LPUART_TransferAbortReceive` (LPUART\_Type \*base, lpuart\_handle\_t \*handle)

*Aborts the interrupt-driven data receiving.*
- `status_t LPUART_TransferGetReceiveCount` (LPUART\_Type \*base, lpuart\_handle\_t \*handle, uint32\_t \*count)

*Gets the number of bytes that have been received.*
- `void LPUART_TransferHandleIRQ` (LPUART\_Type \*base, void \*irqHandle)

*LPUART IRQ handle function.*
- `void LPUART_TransferHandleErrorIRQ` (LPUART\_Type \*base, void \*irqHandle)

*LPUART Error IRQ handle function.*

### 32.2.3 Data Structure Documentation

#### 32.2.3.1 struct lpuart\_config\_t

##### Data Fields

- `uint32_t baudRate_Bps`  
*LPUART baud rate.*
- `lpuart_parity_mode_t parityMode`  
*Parity mode, disabled (default), even, odd.*
- `lpuart_data_bits_t dataBitsCount`  
*Data bits count, eight (default), seven.*
- `bool isMsb`  
*Data bits order, LSB (default), MSB.*
- `lpuart_idle_type_select_t rxIdleType`  
*RX IDLE type.*
- `lpuart_idle_config_t rxIdleConfig`  
*RX IDLE configuration.*
- `bool enableTx`

- *Enable TX.*  
• bool `enableRx`  
*Enable RX.*

### Field Documentation

(1) `lpuart_idle_type_select_t lpuart_config_t::rxIdleType`

(2) `lpuart_idle_config_t lpuart_config_t::rxIdleConfig`

### 32.2.3.2 struct `lpuart_transfer_t`

#### Data Fields

- `size_t dataSize`  
*The byte count to be transfer.*
- `uint8_t * data`  
*The buffer of data to be transfer.*
- `uint8_t * rxData`  
*The buffer to receive data.*
- `const uint8_t * txData`  
*The buffer of data to be sent.*

### Field Documentation

(1) `uint8_t* lpuart_transfer_t::data`

(2) `uint8_t* lpuart_transfer_t::rxData`

(3) `const uint8_t* lpuart_transfer_t::txData`

(4) `size_t lpuart_transfer_t::dataSize`

### 32.2.3.3 struct `_lpuart_handle`

#### Data Fields

- `const uint8_t *volatile txData`  
*Address of remaining data to send.*
- `volatile size_t txDataSize`  
*Size of the remaining data to send.*
- `size_t txDataSizeAll`  
*Size of the data to send out.*
- `uint8_t *volatile rxData`  
*Address of remaining data to receive.*
- `volatile size_t rxDataSize`  
*Size of the remaining data to receive.*
- `size_t rxDataSizeAll`  
*Size of the data to receive.*
- `uint8_t * rxRingBuffer`  
*Start address of the receiver ring buffer.*

- `size_t rxRingBufferSize`  
*Size of the ring buffer.*
- `volatile uint16_t rxRingBufferHead`  
*Index for the driver to store received data into ring buffer.*
- `volatile uint16_t rxRingBufferTail`  
*Index for the user to get data from the ring buffer.*
- `lpuart_transfer_callback_t callback`  
*Callback function.*
- `void * userData`  
*LPUART callback function parameter.*
- `volatile uint8_t txState`  
*TX transfer state.*
- `volatile uint8_t rxState`  
*RX transfer state.*



## Field Documentation

- (1) `const uint8_t* volatile lpuart_handle_t::txData`
- (2) `volatile size_t lpuart_handle_t::txDataSize`
- (3) `size_t lpuart_handle_t::txDataSizeAll`
- (4) `uint8_t* volatile lpuart_handle_t::rxData`
- (5) `volatile size_t lpuart_handle_t::rxDataSize`
- (6) `size_t lpuart_handle_t::rxDataSizeAll`
- (7) `uint8_t* lpuart_handle_t::rxRingBuffer`
- (8) `size_t lpuart_handle_t::rxRingBufferSize`
- (9) `volatile uint16_t lpuart_handle_t::rxRingBufferHead`
- (10) `volatile uint16_t lpuart_handle_t::rxRingBufferTail`
- (11) `lpuart_transfer_callback_t lpuart_handle_t::callback`
- (12) `void* lpuart_handle_t::userData`
- (13) `volatile uint8_t lpuart_handle_t::txState`
- (14) `volatile uint8_t lpuart_handle_t::rxState`

## 32.2.4 Macro Definition Documentation

32.2.4.1 `#define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 7, 3))`

32.2.4.2 `#define UART_RETRY_TIMES 0U /* Defining to zero means to keep waiting for the flag until it is assert/deassert. */`

## 32.2.5 Typedef Documentation

32.2.5.1 `typedef void(* lpuart_transfer_callback_t)(LPUART_Type *base, lpuart_handle_t *handle, status_t status, void *userData)`

## 32.2.6 Enumeration Type Documentation

### 32.2.6.1 anonymous enum

Enumerator

*kStatus\_LPUART\_TxBusy* TX busy.

*kStatus\_LPUART\_RxBusy* RX busy.  
*kStatus\_LPUART\_TxIdle* LPUART transmitter is idle.  
*kStatus\_LPUART\_RxIdle* LPUART receiver is idle.  
*kStatus\_LPUART\_TxWatermarkTooLarge* TX FIFO watermark too large.  
*kStatus\_LPUART\_RxWatermarkTooLarge* RX FIFO watermark too large.  
*kStatus\_LPUART\_FlagCannotClearManually* Some flag can't manually clear.  
*kStatus\_LPUART\_Error* Error happens on LPUART.  
*kStatus\_LPUART\_RxRingBufferOverrun* LPUART RX software ring buffer overrun.  
*kStatus\_LPUART\_RxHardwareOverrun* LPUART RX receiver overrun.  
*kStatus\_LPUART\_NoiseError* LPUART noise error.  
*kStatus\_LPUART\_FramingError* LPUART framing error.  
*kStatus\_LPUART\_ParityError* LPUART parity error.  
*kStatus\_LPUART\_BaudrateNotSupport* Baudrate is not support in current clock source.  
*kStatus\_LPUART\_IdleLineDetected* IDLE flag.  
*kStatus\_LPUART\_Timeout* LPUART times out.

### 32.2.6.2 enum lpuart\_parity\_mode\_t

Enumerator

*kLPUART\_ParityDisabled* Parity disabled.  
*kLPUART\_ParityEven* Parity enabled, type even, bit setting: PE|PT = 10.  
*kLPUART\_ParityOdd* Parity enabled, type odd, bit setting: PE|PT = 11.

### 32.2.6.3 enum lpuart\_data\_bits\_t

Enumerator

*kLPUART\_EightDataBits* Eight data bit.

### 32.2.6.4 enum lpuart\_stop\_bit\_count\_t

Enumerator

*kLPUART\_OneStopBit* One stop bit.  
*kLPUART\_TwoStopBit* Two stop bits.

### 32.2.6.5 enum lpuart\_idle\_type\_select\_t

Enumerator

*kLPUART\_IdleTypeStartBit* Start counting after a valid start bit.  
*kLPUART\_IdleTypeStopBit* Start counting after a stop bit.

### 32.2.6.6 enum lpuart\_idle\_config\_t

This structure defines the number of idle characters that must be received before the IDLE flag is set.

Enumerator

***kLPUART\_IdleCharacter1*** the number of idle characters.  
***kLPUART\_IdleCharacter2*** the number of idle characters.  
***kLPUART\_IdleCharacter4*** the number of idle characters.  
***kLPUART\_IdleCharacter8*** the number of idle characters.  
***kLPUART\_IdleCharacter16*** the number of idle characters.  
***kLPUART\_IdleCharacter32*** the number of idle characters.  
***kLPUART\_IdleCharacter64*** the number of idle characters.  
***kLPUART\_IdleCharacter128*** the number of idle characters.

### 32.2.6.7 enum \_lpuart\_interrupt\_enable

This structure contains the settings for all LPUART interrupt configurations.

Enumerator

***kLPUART\_RxActiveEdgeInterruptEnable*** Receive Active Edge. bit 6  
***kLPUART\_TxDataRegEmptyInterruptEnable*** Transmit data register empty. bit 23  
***kLPUART\_TransmissionCompleteInterruptEnable*** Transmission complete. bit 22  
***kLPUART\_RxDataRegFullInterruptEnable*** Receiver data register full. bit 21  
***kLPUART\_IdleLineInterruptEnable*** Idle line. bit 20  
***kLPUART\_RxOverrunInterruptEnable*** Receiver Overrun. bit 27  
***kLPUART\_NoiseErrorInterruptEnable*** Noise error flag. bit 26  
***kLPUART\_FramingErrorInterruptEnable*** Framing error flag. bit 25  
***kLPUART\_ParityErrorInterruptEnable*** Parity error flag. bit 24

### 32.2.6.8 enum \_lpuart\_flags

This provides constants for the LPUART status flags for use in the LPUART functions.

Enumerator

***kLPUART\_TxDataRegEmptyFlag*** Transmit data register empty flag, sets when transmit buffer is empty. bit 23  
***kLPUART\_TransmissionCompleteFlag*** Transmission complete flag, sets when transmission activity complete. bit 22  
***kLPUART\_RxDataRegFullFlag*** Receive data register full flag, sets when the receive data buffer is full. bit 21  
***kLPUART\_IdleLineFlag*** Idle line detect flag, sets when idle line detected. bit 20



- kLPUART\_RxOverrunFlag*** Receive Overrun, sets when new data is received before data is read from receive register. bit 19
- kLPUART\_NoiseErrorFlag*** Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18
- kLPUART\_FramingErrorFlag*** Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17
- kLPUART\_ParityErrorFlag*** If parity enabled, sets upon parity error detection. bit 16
- kLPUART\_RxActiveEdgeFlag*** Receive pin active edge interrupt flag, sets when active edge detected. bit 30
- kLPUART\_RxActiveFlag*** Receiver Active Flag (RAF), sets at beginning of valid start. bit 24

### 32.2.7 Function Documentation

#### 32.2.7.1 `status_t LPUART_Init ( LPUART_Type * base, const lpuart_config_t * config, uint32_t srcClock_Hz )`

This function configures the LPUART module with user-defined settings. Call the [LPUART\\_GetDefault-Config\(\)](#) function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
* lpuart_config_t lpuartConfig;
* lpuartConfig.baudRate_Bps = 115200U;
* lpuartConfig.parityMode = kLPUART_ParityDisabled;
* lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
* lpuartConfig.isMsb = false;
* lpuartConfig.stopBitCount = kLPUART_OneStopBit;
* lpuartConfig.txFifoWatermark = 0;
* lpuartConfig.rxFifoWatermark = 1;
* LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
*
```

#### Parameters

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>base</i>        | LPUART peripheral base address.                    |
| <i>config</i>      | Pointer to a user-defined configuration structure. |
| <i>srcClock_Hz</i> | LPUART clock source frequency in HZ.               |

#### Return values

|                                           |                                                  |
|-------------------------------------------|--------------------------------------------------|
| <i>kStatus_LPUART_-BaudrateNotSupport</i> | Baudrate is not support in current clock source. |
|-------------------------------------------|--------------------------------------------------|

|                        |                           |
|------------------------|---------------------------|
| <i>kStatus_Success</i> | LPUART initialize succeed |
|------------------------|---------------------------|

### 32.2.7.2 void LPUART\_Deinit ( LPUART\_Type \* *base* )

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

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

### 32.2.7.3 void LPUART\_GetDefaultConfig ( lpuart\_config\_t \* *config* )

This function initializes the LPUART configuration structure to a default value. The default values are:  
 : lpuartConfig->baudRate\_Bps = 115200U; lpuartConfig->parityMode = kLPUART\_ParityDisabled;  
 lpuartConfig->dataBitsCount = kLPUART\_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART\_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART\_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART\_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

Parameters

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

### 32.2.7.4 status\_t LPUART\_SetBaudRate ( LPUART\_Type \* *base*, uint32\_t *baudRate\_Bps*, uint32\_t *srcClock\_Hz* )

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART\_Init.

```
* LPUART_SetBaudRate(LPUART1, 115200U, 200000000U);
*
```

Parameters

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

|                     |                                      |
|---------------------|--------------------------------------|
| <i>baudRate_Bps</i> | LPUART baudrate to be set.           |
| <i>srcClock_Hz</i>  | LPUART clock source frequency in HZ. |

Return values

|                                           |                                                        |
|-------------------------------------------|--------------------------------------------------------|
| <i>kStatus_LPUART_-BaudrateNotSupport</i> | Baudrate is not supported in the current clock source. |
| <i>kStatus_Success</i>                    | Set baudrate succeeded.                                |

### 32.2.7.5 void LPUART\_Enable9bitMode ( LPUART\_Type \* *base*, bool *enable* )

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPUART peripheral base address.   |
| <i>enable</i> | true to enable, false to disable. |

### 32.2.7.6 static void LPUART\_SetMatchAddress ( LPUART\_Type \* *base*, uint16\_t *address1*, uint16\_t *address2* ) [inline], [static]

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receives with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note

Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>base</i>     | LPUART peripheral base address. |
| <i>address1</i> | LPUART slave address1.          |
| <i>address2</i> | LPUART slave address2.          |

### 32.2.7.7 `static void LPUART_EnableMatchAddress ( LPUART_Type * base, bool match1, bool match2 ) [inline], [static]`

Parameters

|               |                                                  |
|---------------|--------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                  |
| <i>match1</i> | true to enable match address1, false to disable. |
| <i>match2</i> | true to enable match address2, false to disable. |

### 32.2.7.8 `uint32_t LPUART_GetStatusFlags ( LPUART_Type * base )`

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators `_lpuart_flags`. To check for a specific status, compare the return value with enumerators in the `_lpuart_flags`. For example, to check whether the TX is empty:

```
* if (kLPUART_TxDataRegEmptyFlag &
* LPUART_GetStatusFlags(LPUART1))
* {
* ...
* }
*
```

Parameters

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

Returns

LPUART status flags which are ORed by the enumerators in the `_lpuart_flags`.

### 32.2.7.9 `status_t LPUART_ClearStatusFlags ( LPUART_Type * base, uint32_t mask )`

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only be cleared or set by hardware are: `kLPUART_TxDataRegEmptyFlag`, `kLPUART_TransmissionCompleteFlag`, `kLPUART_RxDataRegFullFlag`, `kLPUART_RxActiveFlag`, `kLPUART_NoiseErrorFlag`, `kLPUART_ParityErrorFlag`, `kLPUART_TxFifoEmptyFlag`, `kLPUART_RxFifoEmptyFlag`. Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

## Parameters

|             |                                                                                                                                                     |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | LPUART peripheral base address.                                                                                                                     |
| <i>mask</i> | the status flags to be cleared. The user can use the enumerators in the <code>_lpuart_status_flag_t</code> to do the OR operation and get the mask. |

## Returns

0 succeed, others failed.

## Return values

|                                                     |                                                                                         |
|-----------------------------------------------------|-----------------------------------------------------------------------------------------|
| <i>kStatus_LPUART_Flag-<br/>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 are cleared.                                                         |

### 32.2.7.10 void LPUART\_EnableInterrupts ( LPUART\_Type \* *base*, uint32\_t *mask* )

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the [\\_lpuart\\_interrupt\\_enable](#). This examples shows how to enable TX empty interrupt and RX full interrupt:

```
* LPUART_EnableInterrupts(LPUART1,
* kLPUART_TxDataRegEmptyInterruptEnable |
* kLPUART_RxDataRegFullInterruptEnable);
```

## Parameters

|             |                                                                                    |
|-------------|------------------------------------------------------------------------------------|
| <i>base</i> | LPUART peripheral base address.                                                    |
| <i>mask</i> | The interrupts to enable. Logical OR of <a href="#">_lpuart_interrupt_enable</a> . |

### 32.2.7.11 void LPUART\_DisableInterrupts ( LPUART\_Type \* *base*, uint32\_t *mask* )

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See [\\_lpuart\\_interrupt\\_enable](#). This example shows how to disable the TX empty interrupt and RX full interrupt:

```
* LPUART_DisableInterrupts(LPUART1,
* kLPUART_TxDataRegEmptyInterruptEnable |
* kLPUART_RxDataRegFullInterruptEnable);
```

## Parameters

|             |                                                                                     |
|-------------|-------------------------------------------------------------------------------------|
| <i>base</i> | LPUART peripheral base address.                                                     |
| <i>mask</i> | The interrupts to disable. Logical OR of <a href="#">_lpuart_interrupt_enable</a> . |

**32.2.7.12 uint32\_t LPUART\_GetEnabledInterrupts ( LPUART\_Type \* *base* )**

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators [\\_lpuart\\_interrupt\\_enable](#). To check a specific interrupt enable status, compare the return value with enumerators in [\\_lpuart\\_interrupt\\_enable](#). For example, to check whether the TX empty interrupt is enabled:

```
* uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);
*
* if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
* {
* ...
* }
*
```

## Parameters

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

## Returns

LPUART interrupt flags which are logical OR of the enumerators in [\\_lpuart\\_interrupt\\_enable](#).

**32.2.7.13 uint32\_t LPUART\_GetInstance ( LPUART\_Type \* *base* )**

## Parameters

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

## Returns

LPUART instance.

**32.2.7.14 static void LPUART\_EnableTx ( LPUART\_Type \* *base*, bool *enable* )  
[inline], [static]**

This function enables or disables the LPUART transmitter.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPUART peripheral base address.   |
| <i>enable</i> | True to enable, false to disable. |

### 32.2.7.15 static void LPUART\_EnableRx ( LPUART\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the LPUART receiver.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPUART peripheral base address.   |
| <i>enable</i> | True to enable, false to disable. |

### 32.2.7.16 static void LPUART\_WriteByte ( LPUART\_Type \* *base*, uint8\_t *data* ) [inline], [static]

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

## Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | LPUART peripheral base address. |
| <i>data</i> | Data write to the TX register.  |

### 32.2.7.17 static uint8\_t LPUART\_ReadByte ( LPUART\_Type \* *base* ) [inline], [static]

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

## Parameters

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

## Returns

Data read from data register.

### 32.2.7.18 void LPUART\_SendAddress ( LPUART\_Type \* *base*, uint8\_t *address* )

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | LPUART peripheral base address. |
| <i>address</i> | LPUART slave address.           |

**32.2.7.19** `status_t LPUART_WriteBlocking ( LPUART_Type * base, const uint8_t * data, size_t length )`

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the data to be sent out to the bus.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | LPUART peripheral base address.     |
| <i>data</i>   | Start address of the data to write. |
| <i>length</i> | Size of the data to write.          |

Return values

|                               |                                         |
|-------------------------------|-----------------------------------------|
| <i>kStatus_LPUART_Timeout</i> | Transmission timed out and was aborted. |
| <i>kStatus_Success</i>        | Successfully wrote all data.            |

**32.2.7.20** `status_t LPUART_ReadBlocking ( LPUART_Type * base, uint8_t * data, size_t length )`

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                         |
| <i>data</i>   | Start address of the buffer to store the received data. |
| <i>length</i> | Size of the buffer.                                     |

Return values

---



|                                          |                                                 |
|------------------------------------------|-------------------------------------------------|
| <i>kStatus_LPUART_Rx-HardwareOverrun</i> | Receiver overrun happened while receiving data. |
| <i>kStatus_LPUART_Noise-Error</i>        | Noise error happened while receiving data.      |
| <i>kStatus_LPUART_-FramingError</i>      | Framing error happened while receiving data.    |
| <i>kStatus_LPUART_Parity-Error</i>       | Parity error happened while receiving data.     |
| <i>kStatus_LPUART_-Timeout</i>           | Transmission timed out and was aborted.         |
| <i>kStatus_Success</i>                   | Successfully received all data.                 |

### 32.2.7.21 void LPUART\_TransferCreateHandle ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle*, lpuart\_transfer\_callback\_t *callback*, void \* *userData* )

This function initializes the LPUART handle, which can be used for other LPUART transactional APIs. Usually, for a specified LPUART instance, call this API once to get the initialized handle.

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the [LPUART\\_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as *ringBuffer*.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>base</i>     | LPUART peripheral base address. |
| <i>handle</i>   | LPUART handle pointer.          |
| <i>callback</i> | Callback function.              |
| <i>userData</i> | User data.                      |

### 32.2.7.22 status\_t LPUART\_TransferSendNonBlocking ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle*, lpuart\_transfer\_t \* *xfer* )

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the [kStatus\\_LPUART\\_TxIdle](#) as status parameter.

## Note

The `kStatus_LPUART_TxIdle` is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the TX, check the `kLPUART_TransmissionCompleteFlag` to ensure that the transmit is finished.

## Parameters

|               |                                                                    |
|---------------|--------------------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                                    |
| <i>handle</i> | LPUART handle pointer.                                             |
| <i>xfer</i>   | LPUART transfer structure, see <a href="#">lpuart_transfer_t</a> . |

## Return values

|                                |                                                                                    |
|--------------------------------|------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start the data transmission.                                          |
| <i>kStatus_LPUART_TxBusy</i>   | Previous transmission still not finished, data not all written to the TX register. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.                                                                  |

### 32.2.7.23 void LPUART\_TransferStartRingBuffer ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle*, uint8\_t \* *ringBuffer*, size\_t *ringBufferSize* )

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the `UART_TransferReceiveNonBlocking()` API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

## Note

When using RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, then only 31 bytes are used for saving data.

## Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

|                       |                                                                                              |
|-----------------------|----------------------------------------------------------------------------------------------|
| <i>ringBuffer</i>     | Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer. |
| <i>ringBufferSize</i> | size of the ring buffer.                                                                     |

**32.2.7.24 void LPUART\_TransferStopRingBuffer ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle* )**

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

**32.2.7.25 size\_t LPUART\_TransferGetRxRingBufferLength ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle* )**

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

Returns

Length of received data in RX ring buffer.

**32.2.7.26 void LPUART\_TransferAbortSend ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle* )**

This function aborts the interrupt driven data sending. The user can get the remainBbytes to find out how many bytes are not sent out.

Parameters

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

|               |                        |
|---------------|------------------------|
| <i>handle</i> | LPUART handle pointer. |
|---------------|------------------------|

### 32.2.7.27 **status\_t LPUART\_TransferGetSendCount ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle*, uint32\_t \* *count* )**

This function gets the number of bytes that have been sent out to bus by an interrupt method.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |
| <i>count</i>  | Send bytes count.               |

Return values

|                                     |                                                       |
|-------------------------------------|-------------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No send in progress.                                  |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                 |
| <i>kStatus_Success</i>              | Get successfully through the parameter <i>count</i> ; |

### 32.2.7.28 **status\_t LPUART\_TransferReceiveNonBlocking ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle*, lpuart\_transfer\_t \* *xfer*, size\_t \* *receivedBytes* )**

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter *receivedBytes* shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter *kStatus\_UART\_RxIdle*. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to *xfer->data*, which returns with the parameter *receivedBytes* set to 5. For the remaining 5 bytes, the newly arrived data is saved from *xfer->data[5]*. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to *xfer->data*. When all data is received, the upper layer is notified.

Parameters

|                      |                                                               |
|----------------------|---------------------------------------------------------------|
| <i>base</i>          | LPUART peripheral base address.                               |
| <i>handle</i>        | LPUART handle pointer.                                        |
| <i>xfer</i>          | LPUART transfer structure, see <code>uart_transfer_t</code> . |
| <i>receivedBytes</i> | Bytes received from the ring buffer directly.                 |

Return values

|                                |                                                          |
|--------------------------------|----------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully queue the transfer into the transmit queue. |
| <i>kStatus_LPUART_Rx-Busy</i>  | Previous receive request is not finished.                |
| <i>kStatus_InvalidArgument</i> | Invalid argument.                                        |

**32.2.7.29 void LPUART\_TransferAbortReceive ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle* )**

This function aborts the interrupt-driven data receiving. The user can get the `remainBytes` to find out how many bytes not received yet.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

**32.2.7.30 status\_t LPUART\_TransferGetReceiveCount ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle*, uint32\_t \* *count* )**

This function gets the number of bytes that have been received.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |
| <i>count</i>  | Receive bytes count.            |

Return values

|                                     |                                               |
|-------------------------------------|-----------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                       |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                         |
| <i>kStatus_Success</i>              | Get successfully through the parameter count; |

**32.2.7.31 void LPUART\_TransferHandleIRQ ( LPUART\_Type \* *base*, void \* *irqHandle* )**

This function handles the LPUART transmit and receive IRQ request.

Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>base</i>      | LPUART peripheral base address. |
| <i>irqHandle</i> | LPUART handle pointer.          |

**32.2.7.32 void LPUART\_TransferHandleErrorIRQ ( LPUART\_Type \* *base*, void \* *irqHandle* )**

This function handles the LPUART error IRQ request.

Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>base</i>      | LPUART peripheral base address. |
| <i>irqHandle</i> | LPUART handle pointer.          |

## 32.3 LPUART eDMA Driver

### 32.3.1 Overview

#### Data Structures

- struct `lpuart_edma_handle_t`  
*LPUART eDMA handle. [More...](#)*

#### Typedefs

- typedef `void(* lpuart_edma_transfer_callback_t )(LPUART_Type *base, lpuart_edma_handle_t *handle, status_t status, void *userData)`  
*LPUART transfer callback function.*

#### Driver version

- #define `FSL_LPUART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))`  
*LPUART EDMA driver version.*

#### eDMA transactional

- void `LPUART_TransferCreateHandleEDMA (LPUART_Type *base, lpuart_edma_handle_t *handle, lpuart_edma_transfer_callback_t callback, void *userData, edma_handle_t *txEdmaHandle, edma_handle_t *rxEdmaHandle)`  
*Initializes the LPUART handle which is used in transactional functions.*
- `status_t LPUART_SendEDMA (LPUART_Type *base, lpuart_edma_handle_t *handle, lpuart_transfer_t *xfer)`  
*Sends data using eDMA.*
- `status_t LPUART_ReceiveEDMA (LPUART_Type *base, lpuart_edma_handle_t *handle, lpuart_transfer_t *xfer)`  
*Receives data using eDMA.*
- void `LPUART_TransferAbortSendEDMA (LPUART_Type *base, lpuart_edma_handle_t *handle)`  
*Aborts the sent data using eDMA.*
- void `LPUART_TransferAbortReceiveEDMA (LPUART_Type *base, lpuart_edma_handle_t *handle)`  
*Aborts the received data using eDMA.*
- `status_t LPUART_TransferGetSendCountEDMA (LPUART_Type *base, lpuart_edma_handle_t *handle, uint32_t *count)`  
*Gets the number of bytes written to the LPUART TX register.*
- `status_t LPUART_TransferGetReceiveCountEDMA (LPUART_Type *base, lpuart_edma_handle_t *handle, uint32_t *count)`  
*Gets the number of received bytes.*
- void `LPUART_TransferEdmaHandleIRQ (LPUART_Type *base, void *lpuartEdmaHandle)`  
*LPUART eDMA IRQ handle function.*

## 32.3.2 Data Structure Documentation

### 32.3.2.1 struct `_lpuart_edma_handle`

#### Data Fields

- `lpuart_edma_transfer_callback_t` `callback`  
*Callback function.*
- `void *` `userData`  
*LPUART callback function parameter.*
- `size_t` `rxDataSizeAll`  
*Size of the data to receive.*
- `size_t` `txDataSizeAll`  
*Size of the data to send out.*
- `edma_handle_t *` `txEdmaHandle`  
*The eDMA TX channel used.*
- `edma_handle_t *` `rxEdmaHandle`  
*The eDMA RX channel used.*
- `uint8_t` `nbytes`  
*eDMA minor byte transfer count initially configured.*
- `volatile uint8_t` `txState`  
*TX transfer state.*
- `volatile uint8_t` `rxState`  
*RX transfer state.*



## Field Documentation

- (1) `lpuart_edma_transfer_callback_t lpuart_edma_handle_t::callback`
- (2) `void* lpuart_edma_handle_t::userData`
- (3) `size_t lpuart_edma_handle_t::rxDataSizeAll`
- (4) `size_t lpuart_edma_handle_t::txDataSizeAll`
- (5) `edma_handle_t* lpuart_edma_handle_t::txEdmaHandle`
- (6) `edma_handle_t* lpuart_edma_handle_t::rxEdmaHandle`
- (7) `uint8_t lpuart_edma_handle_t::nbytes`
- (8) `volatile uint8_t lpuart_edma_handle_t::txState`

## 32.3.3 Macro Definition Documentation

32.3.3.1 `#define FSL_LPUART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))`

## 32.3.4 Typedef Documentation

32.3.4.1 `typedef void(* lpuart_edma_transfer_callback_t)(LPUART_Type *base, lpuart_edma_handle_t *handle, status_t status, void *userData)`

## 32.3.5 Function Documentation

32.3.5.1 `void LPUART_TransferCreateHandleEDMA ( LPUART_Type * base, lpuart_edma_handle_t * handle, lpuart_edma_transfer_callback_t callback, void * userData, edma_handle_t * txEdmaHandle, edma_handle_t * rxEdmaHandle )`

### Note

This function disables all LPUART interrupts.

### Parameters

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

|                     |                                                         |
|---------------------|---------------------------------------------------------|
| <i>handle</i>       | Pointer to <code>lpuart_edma_handle_t</code> structure. |
| <i>callback</i>     | Callback function.                                      |
| <i>userData</i>     | User data.                                              |
| <i>txEdmaHandle</i> | User requested DMA handle for TX DMA transfer.          |
| <i>rxEdmaHandle</i> | User requested DMA handle for RX DMA transfer.          |

### 32.3.5.2 `status_t LPUART_SendEDMA ( LPUART_Type * base, lpuart_edma_handle_t * handle, lpuart_transfer_t * xfer )`

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

|               |                                                                         |
|---------------|-------------------------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                                         |
| <i>handle</i> | LPUART handle pointer.                                                  |
| <i>xfer</i>   | LPUART eDMA transfer structure. See <a href="#">lpuart_transfer_t</a> . |

Return values

|                                |                             |
|--------------------------------|-----------------------------|
| <i>kStatus_Success</i>         | if succeed, others failed.  |
| <i>kStatus_LPUART_TxBusy</i>   | Previous transfer on going. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.           |

### 32.3.5.3 `status_t LPUART_ReceiveEDMA ( LPUART_Type * base, lpuart_edma_handle_t * handle, lpuart_transfer_t * xfer )`

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                         |
| <i>handle</i> | Pointer to <code>lpuart_edma_handle_t</code> structure. |

|             |                                                                         |
|-------------|-------------------------------------------------------------------------|
| <i>xfer</i> | LPUART eDMA transfer structure, see <a href="#">lpuart_transfer_t</a> . |
|-------------|-------------------------------------------------------------------------|

Return values

|                                |                            |
|--------------------------------|----------------------------|
| <i>kStatus_Success</i>         | if succeed, others fail.   |
| <i>kStatus_LPUART_Rx-Busy</i>  | Previous transfer ongoing. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.          |

#### 32.3.5.4 void LPUART\_TransferAbortSendEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle* )

This function aborts the sent data using eDMA.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.            |
| <i>handle</i> | Pointer to lpuart_edma_handle_t structure. |

#### 32.3.5.5 void LPUART\_TransferAbortReceiveEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle* )

This function aborts the received data using eDMA.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.            |
| <i>handle</i> | Pointer to lpuart_edma_handle_t structure. |

#### 32.3.5.6 status\_t LPUART\_TransferGetSendCountEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |
| <i>count</i>  | Send bytes count.               |

Return values

|                                     |                                                             |
|-------------------------------------|-------------------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No send in progress.                                        |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                       |
| <i>kStatus_Success</i>              | Get successfully through the parameter <code>count</code> ; |

### 32.3.5.7 `status_t LPUART_TransferGetReceiveCountEDMA ( LPUART_Type * base, lpuart_edma_handle_t * handle, uint32_t * count )`

This function gets the number of received bytes.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |
| <i>count</i>  | Receive bytes count.            |

Return values

|                                     |                                                             |
|-------------------------------------|-------------------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                                     |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                       |
| <i>kStatus_Success</i>              | Get successfully through the parameter <code>count</code> ; |

### 32.3.5.8 `void LPUART_TransferEdmaHandleIRQ ( LPUART_Type * base, void * lpuartEdmaHandle )`

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

Note

This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

|                          |                                 |
|--------------------------|---------------------------------|
| <i>base</i>              | LPUART peripheral base address. |
| <i>lpuartEdma-Handle</i> | LPUART handle pointer.          |

## 32.4 LPUART FreeRTOS Driver

### 32.4.1 Overview

#### Data Structures

- struct [lpuart\\_rtos\\_config\\_t](#)  
*LPUART RTOS configuration structure. [More...](#)*

#### Driver version

- #define [FSL\\_LPUART\\_FREERTOS\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 6, 0))  
*LPUART FreeRTOS driver version.*

#### LPUART RTOS Operation

- int [LPUART\\_RTOS\\_Init](#) (lpuart\_rtos\_handle\_t \*handle, lpuart\_handle\_t \*t\_handle, const [lpuart\\_rtos\\_config\\_t](#) \*cfg)  
*Initializes an LPUART instance for operation in RTOS.*
- int [LPUART\\_RTOS\\_Deinit](#) (lpuart\_rtos\_handle\_t \*handle)  
*Deinitializes an LPUART instance for operation.*

#### LPUART transactional Operation

- int [LPUART\\_RTOS\\_Send](#) (lpuart\_rtos\_handle\_t \*handle, uint8\_t \*buffer, uint32\_t length)  
*Sends data in the background.*
- int [LPUART\\_RTOS\\_Receive](#) (lpuart\_rtos\_handle\_t \*handle, uint8\_t \*buffer, uint32\_t length, size\_t \*received)  
*Receives data.*
- int [LPUART\\_RTOS\\_SetRxTimeout](#) (lpuart\_rtos\_handle\_t \*handle, uint32\_t rx\_timeout\_constant\_ms, uint32\_t rx\_timeout\_multiplier\_ms)  
*Set RX timeout in runtime.*
- int [LPUART\\_RTOS\\_SetTxTimeout](#) (lpuart\_rtos\_handle\_t \*handle, uint32\_t tx\_timeout\_constant\_ms, uint32\_t tx\_timeout\_multiplier\_ms)  
*Set TX timeout in runtime.*

### 32.4.2 Data Structure Documentation

#### 32.4.2.1 struct [lpuart\\_rtos\\_config\\_t](#)

##### Data Fields

- LPUART\_Type \* [base](#)  
*UART base address.*

- uint32\_t `srcclk`  
*UART source clock in Hz.*
- uint32\_t `baudrate`  
*Desired communication speed.*
- lpuart\_parity\_mode\_t `parity`  
*Parity setting.*
- lpuart\_stop\_bit\_count\_t `stopbits`  
*Number of stop bits to use.*
- uint8\_t \* `buffer`  
*Buffer for background reception.*
- uint32\_t `buffer_size`  
*Size of buffer for background reception.*
- uint32\_t `rx_timeout_constant_ms`  
*RX timeout applied per receive.*
- uint32\_t `rx_timeout_multiplier_ms`  
*RX timeout added for each byte of the receive.*
- uint32\_t `tx_timeout_constant_ms`  
*TX timeout applied per transmission.*
- uint32\_t `tx_timeout_multiplier_ms`  
*TX timeout added for each byte of the transmission.*

### Field Documentation

- (1) uint32\_t lpuart\_rtos\_config\_t::rx\_timeout\_multiplier\_ms
- (2) uint32\_t lpuart\_rtos\_config\_t::tx\_timeout\_multiplier\_ms

### 32.4.3 Macro Definition Documentation

32.4.3.1 #define FSL\_LPUART\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 6, 0))

### 32.4.4 Function Documentation

32.4.4.1 int LPUART\_RTOS\_Init ( lpuart\_rtos\_handle\_t \* *handle*, lpuart\_handle\_t \* *t\_handle*, const lpuart\_rtos\_config\_t \* *cfg* )

Parameters

|                 |                                                                                    |
|-----------------|------------------------------------------------------------------------------------|
| <i>handle</i>   | The RTOS LPUART handle, the pointer to an allocated space for RTOS context.        |
| <i>t_handle</i> | The pointer to an allocated space to store the transactional layer internal state. |

|            |                                                                                      |
|------------|--------------------------------------------------------------------------------------|
| <i>cfg</i> | The pointer to the parameters required to configure the LPUART after initialization. |
|------------|--------------------------------------------------------------------------------------|

Returns

0 succeed, others failed

#### 32.4.4.2 int LPUART\_RTOS\_Deinit ( lpuart\_rtos\_handle\_t \* *handle* )

This function deinitializes the LPUART module, sets all register value to the reset value, and releases the resources.

Parameters

|               |                         |
|---------------|-------------------------|
| <i>handle</i> | The RTOS LPUART handle. |
|---------------|-------------------------|

#### 32.4.4.3 int LPUART\_RTOS\_Send ( lpuart\_rtos\_handle\_t \* *handle*, uint8\_t \* *buffer*, uint32\_t *length* )

This function sends data. It is an synchronous API. If the hardware buffer is full, the task is in the blocked state.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>handle</i> | The RTOS LPUART handle.        |
| <i>buffer</i> | The pointer to buffer to send. |
| <i>length</i> | The number of bytes to send.   |

#### 32.4.4.4 int LPUART\_RTOS\_Receive ( lpuart\_rtos\_handle\_t \* *handle*, uint8\_t \* *buffer*, uint32\_t *length*, size\_t \* *received* )

This function receives data from LPUART. It is an synchronous API. If any data is immediately available it is returned immediately and the number of bytes received.

Parameters

|               |                         |
|---------------|-------------------------|
| <i>handle</i> | The RTOS LPUART handle. |
|---------------|-------------------------|



|                 |                                                                                  |
|-----------------|----------------------------------------------------------------------------------|
| <i>buffer</i>   | The pointer to buffer where to write received data.                              |
| <i>length</i>   | The number of bytes to receive.                                                  |
| <i>received</i> | The pointer to a variable of size_t where the number of received data is filled. |

**32.4.4.5 int LPUART\_RTOS\_SetRxTimeout ( lpuart\_rtos\_handle\_t \* *handle*, uint32\_t *rx\_timeout\_constant\_ms*, uint32\_t *rx\_timeout\_multiplier\_ms* )**

This function can modify RX timeout between initialization and receive.

param *handle* The RTOS LPUART handle. param *rx\_timeout\_constant\_ms* RX timeout applied per receive. param *rx\_timeout\_multiplier\_ms* RX timeout added for each byte of the receive.

**32.4.4.6 int LPUART\_RTOS\_SetTxTimeout ( lpuart\_rtos\_handle\_t \* *handle*, uint32\_t *tx\_timeout\_constant\_ms*, uint32\_t *tx\_timeout\_multiplier\_ms* )**

This function can modify TX timeout between initialization and send.

param *handle* The RTOS LPUART handle. param *tx\_timeout\_constant\_ms* TX timeout applied per transmission. param *tx\_timeout\_multiplier\_ms* TX timeout added for each byte of the transmission.

## 32.5 LPUART CMSIS Driver

This section describes the programming interface of the LPUART Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method please refer to <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

The LPUART driver includes transactional APIs.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements please write custom code.

### 32.5.1 Function groups

#### 32.5.1.1 LPUART CMSIS GetVersion Operation

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

#### 32.5.1.2 LPUART CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

#### 32.5.1.3 LPUART CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialize the lpuart instance . And this API must be called before you configure a lpuart instance or after you Deinit a lpuart instance.The right steps to start an instance is that you must initialize the instance which been slected 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.

#### 32.5.1.4 LPUART CMSIS Transfer Operation

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

#### 32.5.1.5 LPUART CMSIS Status Operation

This function group gets the LPUART transfer status.

### **32.5.1.6 LPUART CMSIS Control Operation**

This function can configure an instance ,set baudrate for lpuart, get current baudrate ,set transfer data bits and other control command.

## Chapter 33

# OCOTP: On Chip One-Time Programmable controller.

### 33.1 Overview

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

This section contains information describing the requirements for the on-chip eFuse OTP controller along with details about the block functionality and implementation.

### 33.2 OCOTP function group

The OCOTP driver support operating API to allow read and write the fuse map.

#### 33.2.1 Initialization and de-initialization

The function `OCOTP_Init()` is to initialize the OCOTP with peripheral base address and source clock frequency.

The function `OCOTP_Deinit()` is to de-initialize the OCOTP controller with peripheral base address.

#### 33.2.2 Read and Write operation

The function `OCOTP_ReloadShadowRegister()` is to reload the value from the fuse map. this API should be called firstly before reading the register.

The `OCOTP_ReadFuseShadowRegister()` is to read the value from a given address, if operation is success, a known value will be return, othwise, a value of 0xBADABADA will be returned.

The function `OCOTP_WriteFuseShadowRegister()` will write a specific value to a known address. please check the return status o make sure whether the access to register is success.

### 33.3 OCOTP example

This example shows how to get the controller version using API. Due to the eFuse is One-Time programmable, example will only print the information of OCOTP controller version. If more operations are needed, please using the API to implement the write and read operation.

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

## Enumerations

- enum {
  - `kStatus_OCOTP_AccessError` = MAKE\_STATUS(kStatusGroup\_SDK\_OCOTP, 0),
  - `kStatus_OCOTP_CrcFail` = MAKE\_STATUS(kStatusGroup\_SDK\_OCOTP, 1),
  - `kStatus_OCOTP_ReloadError`,
  - `kStatus_OCOTP_ProgramFail` = MAKE\_STATUS(kStatusGroup\_SDK\_OCOTP, 3),
  - `kStatus_OCOTP_Locked` = MAKE\_STATUS(kStatusGroup\_SDK\_OCOTP, 4) }*\_ocotp\_status Error codes for the OCOTP driver.*

## Functions

- void `OCOTP_Init` (OCOTP\_Type \*base, uint32\_t srcClock\_Hz)
  - Initializes OCOTP controller.*
- void `OCOTP_Deinit` (OCOTP\_Type \*base)
  - De-initializes OCOTP controller.*
- static bool `OCOTP_CheckBusyStatus` (OCOTP\_Type \*base)
  - Checking the BUSY bit in CTRL register.*
- static bool `OCOTP_CheckErrorStatus` (OCOTP\_Type \*base)
  - Checking the ERROR bit in CTRL register.*
- static void `OCOTP_ClearErrorStatus` (OCOTP\_Type \*base)
  - Clear the error bit if this bit is set.*
- status\_t `OCOTP_ReloadShadowRegister` (OCOTP\_Type \*base)
  - Reload the shadow register.*
- uint32\_t `OCOTP_ReadFuseShadowRegister` (OCOTP\_Type \*base, uint32\_t address)
  - Read the fuse shadow register with the fuse address.*
- status\_t `OCOTP_ReadFuseShadowRegisterExt` (OCOTP\_Type \*base, uint32\_t address, uint32\_t \*data, uint8\_t fuseWords)
  - Read the fuse shadow register from the fuse address.*
- status\_t `OCOTP_WriteFuseShadowRegister` (OCOTP\_Type \*base, uint32\_t address, uint32\_t data)
  - Write the fuse shadow register with the fuse address and data.*
- status\_t `OCOTP_WriteFuseShadowRegisterWithLock` (OCOTP\_Type \*base, uint32\_t address, uint32\_t data, bool lock)
  - Write the fuse shadow register and lock it.*
- static uint32\_t `OCOTP_GetVersion` (OCOTP\_Type \*base)
  - Get the OCOTP controller version from the register.*

## Driver version

- #define `FSL_OCOTP_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 3))
  - OCOTP driver version.*

## 33.4 Macro Definition Documentation

### 33.4.1 #define FSL\_OCOTP\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 3))

## 33.5 Enumeration Type Documentation

### 33.5.1 anonymous enum

Enumerator

*kStatus\_OCOTP\_AccessError* eFuse and shadow register access error.

*kStatus\_OCOTP\_CrcFail* CRC check failed.

*kStatus\_OCOTP\_ReloadError* Error happens during reload shadow register.

*kStatus\_OCOTP\_ProgramFail* Fuse programming failed.

*kStatus\_OCOTP\_Locked* Fuse is locked and cannot be programmed.

## 33.6 Function Documentation

### 33.6.1 void OCOTP\_Init ( OCOTP\_Type \* *base*, uint32\_t *srcClock\_Hz* )

Parameters

|                    |                                                                                                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | OCOTP peripheral base address.                                                                                                                                                |
| <i>srcClock_Hz</i> | source clock frequency in unit of Hz. When the macro FSL_FEATURE_OCOTP_HAS_TIMING_CTRL is defined as 0, this parameter is not used, application could pass in 0 in this case. |

### 33.6.2 void OCOTP\_Deinit ( OCOTP\_Type \* *base* )

Return values

|                        |                                                    |
|------------------------|----------------------------------------------------|
| <i>kStatus_Success</i> | upon successful execution, error status otherwise. |
|------------------------|----------------------------------------------------|

### 33.6.3 static bool OCOTP\_CheckBusyStatus ( OCOTP\_Type \* *base* ) [inline], [static]

Checking this BUSY bit will help confirm if the OCOTP controller is ready for access.

Parameters

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

Return values

|             |                                    |
|-------------|------------------------------------|
| <i>true</i> | for bit set and false for cleared. |
|-------------|------------------------------------|

### 33.6.4 static bool OCOTP\_CheckErrorStatus ( OCOTP\_Type \* *base* ) [inline], [static]

Parameters

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

Return values

|             |                                    |
|-------------|------------------------------------|
| <i>true</i> | for bit set and false for cleared. |
|-------------|------------------------------------|

### 33.6.5 static void OCOTP\_ClearErrorStatus ( OCOTP\_Type \* *base* ) [inline], [static]

Parameters

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

### 33.6.6 status\_t OCOTP\_ReloadShadowRegister ( OCOTP\_Type \* *base* )

This function will help reload the shadow register without resetting the OCOTP module. Please make sure the OCOTP has been initialized before calling this API.

Parameters

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

Return values

|                        |                 |
|------------------------|-----------------|
| <i>kStatus_Success</i> | Reload success. |
|------------------------|-----------------|

|                                   |                |
|-----------------------------------|----------------|
| <i>kStatus_OCOTP_Reload-Error</i> | Reload failed. |
|-----------------------------------|----------------|

### 33.6.7 uint32\_t OCOTP\_ReadFuseShadowRegister ( OCOTP\_Type \* *base*, uint32\_t *address* )

**Deprecated** Use [OCOTP\\_ReadFuseShadowRegisterExt](#) instead of this function.

Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>base</i>    | OCOTP peripheral base address.    |
| <i>address</i> | the fuse address to be read from. |

Returns

The read out data.

### 33.6.8 status\_t OCOTP\_ReadFuseShadowRegisterExt ( OCOTP\_Type \* *base*, uint32\_t *address*, uint32\_t \* *data*, uint8\_t *fuseWords* )

This function reads fuse from *address*, how many words to read is specified by the parameter *fuseWords*. This function could read at most OCOTP\_READ\_FUSE\_DATA\_COUNT fuse word one time.

Parameters

|                  |                                            |
|------------------|--------------------------------------------|
| <i>base</i>      | OCOTP peripheral base address.             |
| <i>address</i>   | the fuse address to be read from.          |
| <i>data</i>      | Data array to save the readout fuse value. |
| <i>fuseWords</i> | How many words to read.                    |

Return values

|                        |                           |
|------------------------|---------------------------|
| <i>kStatus_Success</i> | Read success.             |
| <i>kStatus_Fail</i>    | Error occurs during read. |

### 33.6.9 status\_t OCOTP\_WriteFuseShadowRegister ( OCOTP\_Type \* *base*, uint32\_t *address*, uint32\_t *data* )

Please make sure the write address is not locked while calling this API.



## Parameters

|                |                                            |
|----------------|--------------------------------------------|
| <i>base</i>    | OCOTP peripheral base address.             |
| <i>address</i> | the fuse address to be written.            |
| <i>data</i>    | the value will be written to fuse address. |

## Return values

|              |                                                                  |
|--------------|------------------------------------------------------------------|
| <i>write</i> | status, kStatus_Success for success and kStatus_Fail for failed. |
|--------------|------------------------------------------------------------------|

### 33.6.10 `status_t OCOTP_WriteFuseShadowRegisterWithLock ( OCOTP_Type * base, uint32_t address, uint32_t data, bool lock )`

Please make sure the write address is not locked while calling this API.

Some OCOTP controller supports ECC mode and redundancy mode (see reference manual for more details). OCOTP controller will auto select ECC or redundancy mode to program the fuse word according to fuse map definition. In ECC mode, the 32 fuse bits in one word can only be written once. In redundancy mode, the word can be written more than once as long as they are different fuse bits. Set parameter `lock` as true to force use ECC mode.

## Parameters

|                |                                                      |
|----------------|------------------------------------------------------|
| <i>base</i>    | OCOTP peripheral base address.                       |
| <i>address</i> | The fuse address to be written.                      |
| <i>data</i>    | The value will be written to fuse address.           |
| <i>lock</i>    | Lock or unlock write fuse shadow register operation. |

## Return values

|                                   |                                                                             |
|-----------------------------------|-----------------------------------------------------------------------------|
| <i>kStatus_Success</i>            | Program and reload success.                                                 |
| <i>kStatus_OCOTP_Locked</i>       | The eFuse word is locked and cannot be programmed.                          |
| <i>kStatus_OCOTP_ProgramFail</i>  | eFuse word programming failed.                                              |
| <i>kStatus_OCOTP_Reload-Error</i> | eFuse word programming success, but error happens during reload the values. |

|                                   |                           |
|-----------------------------------|---------------------------|
| <i>kStatus_OCOTP_Access-Error</i> | Cannot access eFuse word. |
|-----------------------------------|---------------------------|

**33.6.11 static uint32\_t OCOTP\_GetVersion ( OCOTP\_Type \* *base* ) [inline], [static]**

Parameters

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

Return values

|               |                    |
|---------------|--------------------|
| <i>return</i> | the version value. |
|---------------|--------------------|

## Chapter 34

# PIT: Periodic Interrupt Timer

### 34.1 Overview

The MCUXpresso SDK provides a driver for the Periodic Interrupt Timer (PIT) of MCUXpresso SDK devices.

### 34.2 Function groups

The PIT driver supports operating the module as a time counter.

#### 34.2.1 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.

#### 34.2.2 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.

#### 34.2.3 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.

### 34.2.4 Status

Provides functions to get and clear the PIT status.

### 34.2.5 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

## 34.3 Typical use case

### 34.3.1 PIT tick example

Updates the PIT period and toggles an LED periodically. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/pit`

## Data Structures

- struct `pit_config_t`  
*PIT configuration structure. [More...](#)*

## Enumerations

- enum `pit_chnl_t` {  
  `kPIT_Chnl_0` = 0U,  
  `kPIT_Chnl_1`,  
  `kPIT_Chnl_2`,  
  `kPIT_Chnl_3` }
- List of PIT channels.*
- enum `pit_interrupt_enable_t` { `kPIT_TimerInterruptEnable` = `PIT_TCTRL_TIE_MASK` }
- List of PIT interrupts.*
- enum `pit_status_flags_t` { `kPIT_TimerFlag` = `PIT_TFLG_TIF_MASK` }
- List of PIT status flags.*

## Driver version

- #define `FSL_PIT_DRIVER_VERSION` (`MAKE_VERSION`(2, 0, 4))  
*PIT Driver Version 2.0.4.*

## Initialization and deinitialization

- void `PIT_Init` (`PIT_Type` \*base, const `pit_config_t` \*config)  
*Ungates the PIT clock, enables the PIT module, and configures the peripheral for basic operations.*
- void `PIT_Deinit` (`PIT_Type` \*base)  
*Gates the PIT clock and disables the PIT module.*
- static void `PIT_GetDefaultConfig` (`pit_config_t` \*config)  
*Fills in the PIT configuration structure with the default settings.*

## 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.*

## 34.4 Data Structure Documentation

### 34.4.1 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*

## 34.5 Enumeration Type Documentation

### 34.5.1 enum pit\_chnl\_t

## Note

Actual number of available channels is SoC dependent

## Enumerator

- kPIT\_Chnl\_0* PIT channel number 0.
- kPIT\_Chnl\_1* PIT channel number 1.
- kPIT\_Chnl\_2* PIT channel number 2.
- kPIT\_Chnl\_3* PIT channel number 3.

**34.5.2 enum pit\_interrupt\_enable\_t**

## Enumerator

- kPIT\_TimerInterruptEnable* Timer interrupt enable.

**34.5.3 enum pit\_status\_flags\_t**

## Enumerator

- kPIT\_TimerFlag* Timer flag.

**34.6 Function Documentation****34.6.1 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 |

**34.6.2 void PIT\_Deinit ( PIT\_Type \* *base* )**

## Parameters

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

### 34.6.3 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. |
|---------------|-----------------------------------------|

### 34.6.4 static void PIT\_EnableInterrupts ( PIT\_Type \* *base*, pit\_chnl\_t *channel*, uint32\_t *mask* ) [inline], [static]

## Parameters

|                |                                                                                                                     |
|----------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | PIT peripheral base address                                                                                         |
| <i>channel</i> | Timer channel number                                                                                                |
| <i>mask</i>    | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">pit_interrupt_enable_t</a> |

### 34.6.5 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> |
|-------------|----------------------------------------------------------------------------------------------------------------------|

**34.6.6** `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](#)

**34.6.7** `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](#)

**34.6.8** `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> |

### 34.6.9 static void PIT\_SetTimerPeriod ( PIT\_Type \* *base*, pit\_chnl\_t *channel*, uint32\_t *count* ) [inline], [static]

Timers begin counting from the value set by this function until it reaches 0, then it generates an interrupt and load this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

#### Note

Users can call the utility macros provided in `fsl_common.h` to convert to ticks.

#### Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | PIT peripheral base address    |
| <i>channel</i> | Timer channel number           |
| <i>count</i>   | Timer period in units of ticks |

### 34.6.10 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

#### 34.6.11 **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.       |

#### 34.6.12 **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 35

# PMU: Power Management Unit

### 35.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Power Management Unit (PMU) module of MCUXpresso SDK devices. The power management unit (PMU) is designed to simplify the external power interface. The power system can be split into the input power sources and their characteristics, the integrated power transforming and controlling elements, and the final load interconnection and requirements. By using the internal LDO regulators, the number of external supplies is greatly reduced.

The PMU driver provides the APIs to adjust the work condition of each regulator, and can gate the power of some modules.

### Enumerations

- enum {  
    kPMU\_1P1RegulatorOutputOK = (1U << 0U),  
    kPMU\_1P1BrownoutOnOutput = (1U << 1U),  
    kPMU\_3P0RegulatorOutputOK = (1U << 2U),  
    kPMU\_3P0BrownoutOnOutput = (1U << 3U),  
    kPMU\_2P5RegulatorOutputOK = (1U << 4U),  
    kPMU\_2P5BrownoutOnOutput = (1U << 5U) }  
    *PMU Status flags.*
- enum pmu\_1p1\_weak\_reference\_source\_t {  
    kPMU\_1P1WeakReferenceSourceAlt0 = 0U,  
    kPMU\_1P1WeakReferenceSourceAlt1 = 1U }  
    *The source for the reference voltage of the weak 1P1 regulator.*
- enum pmu\_3p0\_vbus\_voltage\_source\_t {  
    kPMU\_3P0VBusVoltageSourceAlt0 = 0U,  
    kPMU\_3P0VBusVoltageSourceAlt1 = 1U }  
    *Input voltage source for LDO\_3P0 from USB VBus.*
- enum pmu\_core\_reg\_voltage\_ramp\_rate\_t {  
    kPMU\_CoreRegVoltageRampRateFast = 0U,  
    kPMU\_CoreRegVoltageRampRateMediumFast = 1U,  
    kPMU\_CoreRegVoltageRampRateMediumSlow = 2U,  
    kPMU\_CoreRegVoltageRampRateSlow = 0U }  
    *Regulator voltage ramp rate.*
- enum pmu\_power\_bandgap\_t {  
    kPMU\_NormalPowerBandgap = 0U,  
    kPMU\_LowPowerBandgap = 1U }  
    *Bandgap select.*

## Driver version

- #define `FSL_PMU_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 1)`)  
*PMU driver version.*

## Status.

- `uint32_t PMU_GetStatusFlags` (`PMU_Type *base`)  
*Get PMU status flags.*

## 1P1 Regular

- static void `PMU_1P1SetWeakReferenceSource` (`PMU_Type *base`, `pmu_1p1_weak_reference_source_t` option)  
*Selects the source for the reference voltage of the weak 1P1 regulator.*
- static void `PMU_1P1EnableWeakRegulator` (`PMU_Type *base`, `bool enable`)  
*Enables the weak 1P1 regulator.*
- static void `PMU_1P1SetRegulatorOutputVoltage` (`PMU_Type *base`, `uint32_t value`)  
*Adjust the 1P1 regulator output voltage.*
- static void `PMU_1P1SetBrownoutOffsetVoltage` (`PMU_Type *base`, `uint32_t value`)  
*Adjust the 1P1 regulator brownout offset voltage.*
- static void `PMU_1P1EnablePullDown` (`PMU_Type *base`, `bool enable`)  
*Enable the pull-down circuitry in the regulator.*
- static void `PMU_1P1EnableCurrentLimit` (`PMU_Type *base`, `bool enable`)  
*Enable the current-limit circuitry in the regulator.*
- static void `PMU_1P1EnableBrownout` (`PMU_Type *base`, `bool enable`)  
*Enable the brownout circuitry in the regulator.*
- static void `PMU_1P1EnableOutput` (`PMU_Type *base`, `bool enable`)  
*Enable the regulator output.*

## 3P0 Regular

- static void `PMU_3P0SetRegulatorOutputVoltage` (`PMU_Type *base`, `uint32_t value`)  
*Adjust the 3P0 regulator output voltage.*
- static void `PMU_3P0SetVBusVoltageSource` (`PMU_Type *base`, `pmu_3p0_vbus_voltage_source_t` option)  
*Select input voltage source for LDO\_3P0.*
- static void `PMU_3P0SetBrownoutOffsetVoltage` (`PMU_Type *base`, `uint32_t value`)  
*Adjust the 3P0 regulator brownout offset voltage.*
- static void `PMU_3P0EnableCurrentLimit` (`PMU_Type *base`, `bool enable`)  
*Enable the current-limit circuitry in the 3P0 regulator.*
- static void `PMU_3P0EnableBrownout` (`PMU_Type *base`, `bool enable`)  
*Enable the brownout circuitry in the 3P0 regulator.*
- static void `PMU_3P0EnableOutput` (`PMU_Type *base`, `bool enable`)  
*Enable the 3P0 regulator output.*

## 2P5 Regulator

- static void `PMU_2P5EnableWeakRegulator` (`PMU_Type *base`, `bool enable`)  
*Enables the weak 2P5 regulator.*
- static void `PMU_2P5SetRegulatorOutputVoltage` (`PMU_Type *base`, `uint32_t value`)

- *Adjust the 1P1 regulator output voltage.*  
static void [PMU\\_2P5SetBrownoutOffsetVoltage](#) (PMU\_Type \*base, uint32\_t value)
- *Adjust the 2P5 regulator brownout offset voltage.*  
static void [PMU\\_2P5EnablePullDown](#) (PMU\_Type \*base, bool enable)
- *Enable the pull-down circuitry in the 2P5 regulator.*  
static void [PMU\\_2P1EnablePullDown](#) (PMU\_Type \*base, bool enable)
- *Enable the pull-down circuitry in the 2P5 regulator.*  
static void [PMU\\_2P5EnableCurrentLimit](#) (PMU\_Type \*base, bool enable)
- *Enable the current-limit circuitry in the 2P5 regulator.*  
static void [PMU\\_2P5enableBrownout](#) (PMU\_Type \*base, bool enable)
- *Enable the brownout circuitry in the 2P5 regulator.*  
static void [PMU\\_2P5EnableOutput](#) (PMU\_Type \*base, bool enable)
- *Enable the 2P5 regulator output.*

## Core Regulator

- static void [PMU\\_CoreEnableIncreaseGateDrive](#) (PMU\_Type \*base, bool enable)  
*Increase the gate drive on power gating FETs.*
- static void [PMU\\_CoreSetRegulatorVoltageRampRate](#) (PMU\_Type \*base, [pmu\\_core\\_reg\\_voltage\\_ramp\\_rate\\_t](#) option)  
*Set the CORE regulator voltage ramp rate.*
- static void [PMU\\_CoreSetSOCDomainVoltage](#) (PMU\_Type \*base, uint32\_t value)  
*Define the target voltage for the SOC power domain.*
- static void [PMU\\_CoreSetARMCoreDomainVoltage](#) (PMU\_Type \*base, uint32\_t value)  
*Define the target voltage for the ARM Core power domain.*

## 35.2 Macro Definition Documentation

### 35.2.1 #define FSL\_PMU\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))

Version 2.1.1.

## 35.3 Enumeration Type Documentation

### 35.3.1 anonymous enum

Enumerator

- ***kPMU\_1P1RegulatorOutputOK*** Status bit that signals when the 1p1 regulator output is ok. 1 = regulator output > brownout target.
- ***kPMU\_1P1BrownoutOnOutput*** Status bit that signals when a 1p1 brownout is detected on the regulator output.
- ***kPMU\_3P0RegulatorOutputOK*** Status bit that signals when the 3p0 regulator output is ok. 1 = regulator output > brownout target.
- ***kPMU\_3P0BrownoutOnOutput*** Status bit that signals when a 3p0 brownout is detected on the regulator output.
- ***kPMU\_2P5RegulatorOutputOK*** Status bit that signals when the 2p5 regulator output is ok. 1 = regulator output > brownout target.

*kPMU\_2P5BrownoutOnOutput* Status bit that signals when a 2p5 brownout is detected on the regulator output.

### 35.3.2 enum pmu\_1p1\_weak\_reference\_source\_t

Enumerator

*kPMU\_1P1WeakReferenceSourceAlt0* Weak-linreg output tracks low-power-bandgap voltage.

*kPMU\_1P1WeakReferenceSourceAlt1* Weak-linreg output tracks VDD\_SOC\_CAP voltage.

### 35.3.3 enum pmu\_3p0\_vbus\_voltage\_source\_t

Enumerator

*kPMU\_3P0VBusVoltageSourceAlt0* USB\_OTG1\_VBUS - Utilize VBUS OTG1 for power.

*kPMU\_3P0VBusVoltageSourceAlt1* USB\_OTG2\_VBUS - Utilize VBUS OTG2 for power.

### 35.3.4 enum pmu\_core\_reg\_voltage\_ramp\_rate\_t

Enumerator

*kPMU\_CoreRegVoltageRampRateFast* Fast.

*kPMU\_CoreRegVoltageRampRateMediumFast* Medium Fast.

*kPMU\_CoreRegVoltageRampRateMediumSlow* Medium Slow.

*kPMU\_CoreRegVoltageRampRateSlow* Slow.

### 35.3.5 enum pmu\_power\_bandgap\_t

Enumerator

*kPMU\_NormalPowerBandgap* Normal power bandgap.

*kPMU\_LowPowerBandgap* Low power bandgap.

## 35.4 Function Documentation

### 35.4.1 uint32\_t PMU\_GetStatusFlags ( PMU\_Type \* base )

## Parameters

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

## Returns

PMU status flags. It indicates if regulator output of 1P1, 3P0 and 2P5 is ok and brownout output of 1P1, 3P0 and 2P5 is detected.

### 35.4.2 static void PMU\_1P1SetWeakReferenceSource ( PMU\_Type \* *base*, pmu\_1p1\_weak\_reference\_source\_t *option* ) [inline], [static]

## Parameters

|               |                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------|
| <i>base</i>   | PMU peripheral base address.                                                                      |
| <i>option</i> | The option for reference voltage source, see to <a href="#">pmu_1p1_weak_reference_source_t</a> . |

### 35.4.3 static void PMU\_1P1EnableWeakRegulator ( PMU\_Type \* *base*, bool *enable* ) [inline], [static]

This regulator can be used when the main 1P1 regulator is disabled, under low-power conditions.

## Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PMU peripheral base address. |
| <i>enable</i> | Enable the feature or not.   |

### 35.4.4 static void PMU\_1P1SetRegulatorOutputVoltage ( PMU\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Each LSB is worth 25mV. Programming examples are detailed below. Other output target voltages may be interpolated from these examples. Choices must be in this range:

- 0x1b(1.375V) >= output\_trg >= 0x04(0.8V)
- 0x04 : 0.8V
- 0x10 : 1.1V (typical)
- 0x1b : 1.375V NOTE: There may be reduced chip functionality or reliability at the extremes of the programming range.

Parameters

|              |                               |
|--------------|-------------------------------|
| <i>base</i>  | PMU peripheral base address.  |
| <i>value</i> | Setting value for the output. |

#### 35.4.5 static void PMU\_1P1SetBrownoutOffsetVoltage ( PMU\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Control bits to adjust the regulator brownout offset voltage in 25mV steps. The reset brown-offset is 175mV below the programmed target code. Brownout target = OUTPUT\_TRG - BO\_OFFSET. Some steps may be irrelevant because of input supply limitations or load operation.

Parameters

|              |                                                                         |
|--------------|-------------------------------------------------------------------------|
| <i>base</i>  | PMU peripheral base address.                                            |
| <i>value</i> | Setting value for the brownout offset. The available range is in 3-bit. |

#### 35.4.6 static void PMU\_1P1EnablePullDown ( PMU\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PMU peripheral base address. |
| <i>enable</i> | Enable the feature or not.   |

#### 35.4.7 static void PMU\_1P1EnableCurrentLimit ( PMU\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PMU peripheral base address. |
| <i>enable</i> | Enable the feature or not.   |

#### 35.4.8 static void PMU\_1P1EnableBrownout ( PMU\_Type \* *base*, bool *enable* ) [inline], [static]



Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PMU peripheral base address. |
| <i>enable</i> | Enable the feature or not.   |

### 35.4.9 static void PMU\_1P1EnableOutput ( PMU\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PMU peripheral base address. |
| <i>enable</i> | Enable the feature or not.   |

### 35.4.10 static void PMU\_3P0SetRegulatorOutputVoltage ( PMU\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Each LSB is worth 25mV. Programming examples are detailed below. Other output target voltages may be interpolated from these examples. Choices must be in this range:

- 0x00(2.625V) >= output\_trg >= 0x1f(3.4V)
- 0x00 : 2.625V
- 0x0f : 3.0V (typical)
- 0x1f : 3.4V

Parameters

|              |                               |
|--------------|-------------------------------|
| <i>base</i>  | PMU peripheral base address.  |
| <i>value</i> | Setting value for the output. |

### 35.4.11 static void PMU\_3P0SetVBusVoltageSource ( PMU\_Type \* *base*, pmu\_3p0\_vbus\_voltage\_source\_t *option* ) [inline], [static]

Select input voltage source for LDO\_3P0 from either USB\_OTG1\_VBUS or USB\_OTG2\_VBUS. If only one of the two VBUS voltages is present, it is automatically selected.

Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | PMU peripheral base address.                   |
| <i>option</i> | User-defined input voltage source for LDO_3P0. |

#### 35.4.12 **static void PMU\_3P0SetBrownoutOffsetVoltage ( PMU\_Type \* *base*, uint32\_t *value* ) [inline], [static]**

Control bits to adjust the 3P0 regulator brownout offset voltage in 25mV steps. The reset brown-offset is 175mV below the programmed target code. Brownout target = OUTPUT\_TRG - BO\_OFFSET. Some steps may be irrelevant because of input supply limitations or load operation.

Parameters

|              |                                                                         |
|--------------|-------------------------------------------------------------------------|
| <i>base</i>  | PMU peripheral base address.                                            |
| <i>value</i> | Setting value for the brownout offset. The available range is in 3-bit. |

#### 35.4.13 **static void PMU\_3P0EnableCurrentLimit ( PMU\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PMU peripheral base address. |
| <i>enable</i> | Enable the feature or not.   |

#### 35.4.14 **static void PMU\_3P0EnableBrownout ( PMU\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PMU peripheral base address. |
| <i>enable</i> | Enable the feature or not.   |

#### 35.4.15 **static void PMU\_3P0EnableOutput ( PMU\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PMU peripheral base address. |
| <i>enable</i> | Enable the feature or not.   |

#### 35.4.16 **static void PMU\_2P5EnableWeakRegulator ( PMU\_Type \* *base*, bool *enable* ) [inline], [static]**

This low power regulator is used when the main 2P5 regulator is disabled to keep the 2.5V output roughly at 2.5V. Scales directly with the value of VDDHIGH\_IN.

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PMU peripheral base address. |
| <i>enable</i> | Enable the feature or not.   |

#### 35.4.17 **static void PMU\_2P5SetRegulatorOutputVoltage ( PMU\_Type \* *base*, uint32\_t *value* ) [inline], [static]**

Each LSB is worth 25mV. Programming examples are detailed below. Other output target voltages may be interpolated from these examples. Choices must be in this range:

- 0x00(2.1V) >= output\_trg >= 0x1f(2.875V)
- 0x00 : 2.1V
- 0x10 : 2.5V (typical)
- 0x1f : 2.875V NOTE: There may be reduced chip functionality or reliability at the extremes of the programming range.

Parameters

|              |                               |
|--------------|-------------------------------|
| <i>base</i>  | PMU peripheral base address.  |
| <i>value</i> | Setting value for the output. |

#### 35.4.18 **static void PMU\_2P5SetBrownoutOffsetVoltage ( PMU\_Type \* *base*, uint32\_t *value* ) [inline], [static]**

Adjust the regulator brownout offset voltage in 25mV steps. The reset brown-offset is 175mV below the programmed target code. Brownout target = OUTPUT\_TRG - BO\_OFFSET. Some steps may be irrelevant because of input supply limitations or load operation.

Parameters

|              |                                                                         |
|--------------|-------------------------------------------------------------------------|
| <i>base</i>  | PMU peripheral base address.                                            |
| <i>value</i> | Setting value for the brownout offset. The available range is in 3-bit. |

**35.4.19** `static void PMU_2P5EnablePullDown ( PMU_Type * base, bool enable )`  
`[inline], [static]`

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PMU peripheral base address. |
| <i>enable</i> | Enable the feature or not.   |

**35.4.20** `static void PMU_2P1EnablePullDown ( PMU_Type * base, bool enable )`  
`[inline], [static]`

**Deprecated** Do not use this function. It has been superseded by [PMU\\_2P5EnablePullDown](#).

**35.4.21** `static void PMU_2P5EnableCurrentLimit ( PMU_Type * base, bool enable )`  
`[inline], [static]`

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PMU peripheral base address. |
| <i>enable</i> | Enable the feature or not.   |

**35.4.22** `static void PMU_2P5nableBrownout ( PMU_Type * base, bool enable )`  
`[inline], [static]`

Parameters

---

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PMU peripheral base address. |
| <i>enable</i> | Enable the feature or not.   |

### 35.4.23 `static void PMU_2P5EnableOutput ( PMU_Type * base, bool enable ) [inline], [static]`

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PMU peripheral base address. |
| <i>enable</i> | Enable the feature or not.   |

### 35.4.24 `static void PMU_CoreEnableIncreaseGateDrive ( PMU_Type * base, bool enable ) [inline], [static]`

If set, increases the gate drive on power gating FETs to reduce leakage in the off state. Care must be taken to apply this bit only when the input supply voltage to the power FET is less than 1.1V. NOTE: This bit should only be used in low-power modes where the external input supply voltage is nominally 0.9V.

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PMU peripheral base address. |
| <i>enable</i> | Enable the feature or not.   |

### 35.4.25 `static void PMU_CoreSetRegulatorVoltageRampRate ( PMU_Type * base, pmu_core_reg_voltage_ramp_rate_t option ) [inline], [static]`

Parameters

|               |                                                                                                      |
|---------------|------------------------------------------------------------------------------------------------------|
| <i>base</i>   | PMU peripheral base address.                                                                         |
| <i>option</i> | User-defined option for voltage ramp rate, see to <a href="#">pmu_core_reg_voltage_ramp_rate_t</a> . |

### 35.4.26 `static void PMU_CoreSetSOCDomainVoltage ( PMU_Type * base, uint32_t value ) [inline], [static]`

Define the target voltage for the SOC power domain. Single-bit increments reflect 25mV core voltage steps. Some steps may not be relevant because of input supply limitations or load operation.

- 0x00 : Power gated off.
- 0x01 : Target core voltage = 0.725V
- 0x02 : Target core voltage = 0.750V
- ...
- 0x10 : Target core voltage = 1.100V
- ...
- 0x1e : Target core voltage = 1.450V
- 0x1F : Power FET switched full on. No regulation. NOTE: This register is capable of programming an over-voltage condition on the device. Consult the datasheet Operating Ranges table for the allowed voltages.

Parameters

|              |                                                   |
|--------------|---------------------------------------------------|
| <i>base</i>  | PMU peripheral base address.                      |
| <i>value</i> | Setting value for target voltage. 5-bit available |

### 35.4.27 `static void PMU_CoreSetARMCoreDomainVoltage ( PMU_Type * base, uint32_t value ) [inline], [static]`

Define the target voltage for the ARM Core power domain. Single-bit increments reflect 25mV core voltage steps. Some steps may not be relevant because of input supply limitations or load operation.

- 0x00 : Power gated off.
- 0x01 : Target core voltage = 0.725V
- 0x02 : Target core voltage = 0.750V
- ...
- 0x10 : Target core voltage = 1.100V
- ...
- 0x1e : Target core voltage = 1.450V
- 0x1F : Power FET switched full on. No regulation. NOTE: This register is capable of programming an over-voltage condition on the device. Consult the datasheet Operating Ranges table for the allowed voltages.

## Parameters

|              |                                                   |
|--------------|---------------------------------------------------|
| <i>base</i>  | PMU peripheral base address.                      |
| <i>value</i> | Setting value for target voltage. 5-bit available |

## Chapter 36

# PWM: Pulse Width Modulator

### 36.1 Overview

The MCUXpresso SDK provides a driver for the Pulse Width Modulator (PWM) of MCUXpresso SDK devices.

### 36.2 PWM: Pulse Width Modulator

#### 36.2.1 Initialization and deinitialization

The function [PWM\\_Init\(\)](#) initializes the PWM sub module with specified configurations, the function [PWM\\_GetDefaultConfig\(\)](#) could help to get the default configurations. The initialization function configures the sub module for the requested register update mode for registers with buffers. It also sets up the sub module operation in debug and wait modes.

#### 36.2.2 PWM Operations

The function [PWM\\_SetupPwm\(\)](#) sets up PWM channels for PWM output, the function can set up PWM signal properties for multiple channels. The PWM has 2 channels: A and B. 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 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 also sets up the channel dead time value which is used when the user selects complementary mode of operation.

The function [PWM\\_UpdatePwmDutycycle\(\)](#) updates the PWM signal duty cycle of a particular PWM channel.

#### 36.2.3 Input capture operations

The function [PWM\\_SetupInputCapture\(\)](#) sets up a PWM channel for input capture. The user can specify the capture edge and the mode; one-shot capture or free-running capture.

#### 36.2.4 Fault operation

The function [PWM\\_SetupFault\(\)](#) sets up the properties for each fault.



### 36.2.5 PWM Start and Stop operations

The function `PWM_StartTimer()` can be used to start one or multiple sub modules. The function `PWM_StopTimer()` can be used to stop one or multiple sub modules.

### 36.2.6 Status

Provide functions to get and clear the PWM status.

### 36.2.7 Interrupt

Provide functions to enable/disable PWM interrupts and get current enabled interrupts.

## 36.3 Register Update

Some of the PWM registers have buffers, the driver support various methods to update these registers with the content of the register buffer. 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/pwm` The user can select one of the reload options provided in enumeration `pwm_register_reload_t`. When using immediate reload, the `reloadFrequency` field is not used.

The driver initialization function sets up the appropriate bits in the PWM module based on the register update options selected.

The below function should be used to initiate a register reload. The example shows register reload initiated on PWM sub modules 0, 1, and 2. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/pwm`

## 36.4 Typical use case

### 36.4.1 PWM output

Output PWM signal on 3 PWM sub module with different dutycycles. Periodically update the PWM signal duty cycle. Each sub module runs in Complementary output mode with PWM A used to generate the complementary PWM pair. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/pwm`

## Data Structures

- struct `pwm_signal_param_t`  
*Structure for the user to define the PWM signal characteristics. [More...](#)*
- struct `pwm_config_t`  
*PWM config structure. [More...](#)*
- struct `pwm_fault_input_filter_param_t`

- Structure for the user to configure the fault input filter. [More...](#)
- struct `pwm_fault_param_t`  
Structure is used to hold the parameters to configure a PWM fault. [More...](#)
- struct `pwm_input_capture_param_t`  
Structure is used to hold parameters to configure the capture capability of a signal pin. [More...](#)

## Macros

- #define `PWM_SUBMODULE_SWCONTROL_WIDTH` 2  
Number of bits per submodule for software output control.
- #define `PWM_SUBMODULE_CHANNEL` 2  
Because setting the pwm duty cycle doesn't support PWMX, getting the pwm duty cycle also doesn't support PWMX.

## Enumerations

- enum `pwm_submodule_t` {  
  `kPWM_Module_0` = 0U,  
  `kPWM_Module_1`,  
  `kPWM_Module_2`,  
  `kPWM_Module_3` }  
List of PWM submodules.
- enum `pwm_channels_t`  
List of PWM channels in each module.
- enum `pwm_value_register_t` {  
  `kPWM_ValueRegister_0` = 0U,  
  `kPWM_ValueRegister_1`,  
  `kPWM_ValueRegister_2`,  
  `kPWM_ValueRegister_3`,  
  `kPWM_ValueRegister_4`,  
  `kPWM_ValueRegister_5` }  
List of PWM value registers.
- enum `_pwm_value_register_mask` {  
  `kPWM_ValueRegisterMask_0` = (1U << 0),  
  `kPWM_ValueRegisterMask_1` = (1U << 1),  
  `kPWM_ValueRegisterMask_2` = (1U << 2),  
  `kPWM_ValueRegisterMask_3` = (1U << 3),  
  `kPWM_ValueRegisterMask_4` = (1U << 4),  
  `kPWM_ValueRegisterMask_5` = (1U << 5) }  
List of PWM value registers mask.
- enum `pwm_clock_source_t` {  
  `kPWM_BusClock` = 0U,  
  `kPWM_ExternalClock`,  
  `kPWM_Submodule0Clock` }  
PWM clock source selection.
- enum `pwm_clock_prescale_t` {

```

kPWM_Prescale_Divide_1 = 0U,
kPWM_Prescale_Divide_2,
kPWM_Prescale_Divide_4,
kPWM_Prescale_Divide_8,
kPWM_Prescale_Divide_16,
kPWM_Prescale_Divide_32,
kPWM_Prescale_Divide_64,
kPWM_Prescale_Divide_128 }

```

*PWM prescaler factor selection for clock source.*

- enum `pwm_force_output_trigger_t` {
 

```

kPWM_Force_Local = 0U,
kPWM_Force_Master,
kPWM_Force_LocalReload,
kPWM_Force_MasterReload,
kPWM_Force_LocalSync,
kPWM_Force_MasterSync,
kPWM_Force_External,
kPWM_Force_ExternalSync }

```

*Options that can trigger a PWM FORCE\_OUT.*

- enum `pwm_output_state_t` {
 

```

kPWM_HighState = 0,
kPWM_LowState,
kPWM_NormalState,
kPWM_InvertState,
kPWM_MaskState }

```

*PWM channel output status.*

- enum `pwm_init_source_t` {
 

```

kPWM_Initialize_LocalSync = 0U,
kPWM_Initialize_MasterReload,
kPWM_Initialize_MasterSync,
kPWM_Initialize_ExtSync }

```

*PWM counter initialization options.*

- enum `pwm_load_frequency_t` {

- ```

kPWM_LoadEveryOportunity = 0U,
kPWM_LoadEvery2Oportunity,
kPWM_LoadEvery3Oportunity,
kPWM_LoadEvery4Oportunity,
kPWM_LoadEvery5Oportunity,
kPWM_LoadEvery6Oportunity,
kPWM_LoadEvery7Oportunity,
kPWM_LoadEvery8Oportunity,
kPWM_LoadEvery9Oportunity,
kPWM_LoadEvery10Oportunity,
kPWM_LoadEvery11Oportunity,
kPWM_LoadEvery12Oportunity,
kPWM_LoadEvery13Oportunity,
kPWM_LoadEvery14Oportunity,
kPWM_LoadEvery15Oportunity,
kPWM_LoadEvery16Oportunity }

```
- PWM load frequency selection.*
- enum `pwm_fault_input_t` {

```

kPWM_Fault_0 = 0U,
kPWM_Fault_1,
kPWM_Fault_2,
kPWM_Fault_3 }

```

List of PWM fault selections.
 - enum `pwm_fault_disable_t` {

```

kPWM_FaultDisable_0 = (1U << 0),
kPWM_FaultDisable_1 = (1U << 1),
kPWM_FaultDisable_2 = (1U << 2),
kPWM_FaultDisable_3 = (1U << 3) }

```

List of PWM fault disable mapping selections.
 - enum `pwm_fault_channels_t`

List of PWM fault channels.
 - enum `pwm_input_capture_edge_t` {

```

kPWM_Disable = 0U,
kPWM_FallingEdge,
kPWM_RisingEdge,
kPWM_RiseAndFallEdge }

```

PWM capture edge select.
 - enum `pwm_force_signal_t` {

```

kPWM_UsePwm = 0U,
kPWM_InvertedPwm,
kPWM_SoftwareControl,
kPWM_UseExternal }

```

PWM output options when a FORCE_OUT signal is asserted.
 - enum `pwm_chnl_pair_operation_t` {

```

kPWM_Independent = 0U,
kPWM_ComplementaryPwmA,

```

```
kPWM_ComplementaryPwmB }
```

Options available for the PWM A & B pair operation.

- enum `pwm_register_reload_t` {
`kPWM_ReloadImmediate = 0U`,
`kPWM_ReloadPwmHalfCycle`,
`kPWM_ReloadPwmFullCycle`,
`kPWM_ReloadPwmHalfAndFullCycle` }

Options available on how to load the buffered-registers with new values.

- enum `pwm_fault_recovery_mode_t` {
`kPWM_NoRecovery = 0U`,
`kPWM_RecoverHalfCycle`,
`kPWM_RecoverFullCycle`,
`kPWM_RecoverHalfAndFullCycle` }

Options available on how to re-enable the PWM output when recovering from a fault.

- enum `pwm_interrupt_enable_t` {
`kPWM_CompareVal0InterruptEnable = (1U << 0)`,
`kPWM_CompareVal1InterruptEnable = (1U << 1)`,
`kPWM_CompareVal2InterruptEnable = (1U << 2)`,
`kPWM_CompareVal3InterruptEnable = (1U << 3)`,
`kPWM_CompareVal4InterruptEnable = (1U << 4)`,
`kPWM_CompareVal5InterruptEnable = (1U << 5)`,
`kPWM_CaptureX0InterruptEnable = (1U << 6)`,
`kPWM_CaptureX1InterruptEnable = (1U << 7)`,
`kPWM_CaptureB0InterruptEnable = (1U << 8)`,
`kPWM_CaptureB1InterruptEnable = (1U << 9)`,
`kPWM_CaptureA0InterruptEnable = (1U << 10)`,
`kPWM_CaptureA1InterruptEnable = (1U << 11)`,
`kPWM_ReloadInterruptEnable = (1U << 12)`,
`kPWM_ReloadErrorInterruptEnable = (1U << 13)`,
`kPWM_Fault0InterruptEnable = (1U << 16)`,
`kPWM_Fault1InterruptEnable = (1U << 17)`,
`kPWM_Fault2InterruptEnable = (1U << 18)`,
`kPWM_Fault3InterruptEnable = (1U << 19)` }

List of PWM interrupt options.

- enum `pwm_status_flags_t` {

```

kPWM_CompareVal0Flag = (1U << 0),
kPWM_CompareVal1Flag = (1U << 1),
kPWM_CompareVal2Flag = (1U << 2),
kPWM_CompareVal3Flag = (1U << 3),
kPWM_CompareVal4Flag = (1U << 4),
kPWM_CompareVal5Flag = (1U << 5),
kPWM_CaptureX0Flag = (1U << 6),
kPWM_CaptureX1Flag = (1U << 7),
kPWM_CaptureB0Flag = (1U << 8),
kPWM_CaptureB1Flag = (1U << 9),
kPWM_CaptureA0Flag = (1U << 10),
kPWM_CaptureA1Flag = (1U << 11),
kPWM_ReloadFlag = (1U << 12),
kPWM_ReloadErrorFlag = (1U << 13),
kPWM_RegUpdatedFlag = (1U << 14),
kPWM_Fault0Flag = (1U << 16),
kPWM_Fault1Flag = (1U << 17),
kPWM_Fault2Flag = (1U << 18),
kPWM_Fault3Flag = (1U << 19) }

```

List of PWM status flags.

- enum pwm_dma_enable_t {


```

kPWM_CaptureX0DMAEnable = (1U << 0),
kPWM_CaptureX1DMAEnable = (1U << 1),
kPWM_CaptureB0DMAEnable = (1U << 2),
kPWM_CaptureB1DMAEnable = (1U << 3),
kPWM_CaptureA0DMAEnable = (1U << 4),
kPWM_CaptureA1DMAEnable = (1U << 5) }

```

List of PWM DMA options.

- enum pwm_dma_source_select_t {


```

kPWM_DMAResourceDisable = 0U,
kPWM_DMAWatermarksEnable,
kPWM_DMALocalSync,
kPWM_DMALocalReload }

```

List of PWM capture DMA enable source select.

- enum pwm_watermark_control_t {


```

kPWM_FIFOWatermarksOR = 0U,
kPWM_FIFOWatermarksAND }

```

PWM FIFO Watermark AND Control.

- enum pwm_mode_t {


```

kPWM_SignedCenterAligned = 0U,
kPWM_CenterAligned,
kPWM_SignedEdgeAligned,
kPWM_EdgeAligned }

```

PWM operation mode.

- enum pwm_level_select_t {


```

kPWM_HighTrue = 0U,

```

- ```
kPWM_LowTrue }
 PWM output pulse mode, high-true or low-true.
```
- enum `pwm_fault_state_t` {  
`kPWM_PwmFaultState0`,  
`kPWM_PwmFaultState1`,  
`kPWM_PwmFaultState2`,  
`kPWM_PwmFaultState3` }  
*PWM output fault status.*
  - enum `pwm_reload_source_select_t` {  
`kPWM_LocalReload = 0U`,  
`kPWM_MasterReload` }  
*PWM reload source select.*
  - enum `pwm_fault_clear_t` {  
`kPWM_Automatic = 0U`,  
`kPWM_ManualNormal`,  
`kPWM_ManualSafety` }  
*PWM fault clearing options.*
  - enum `pwm_module_control_t` {  
`kPWM_Control_Module_0 = (1U << 0)`,  
`kPWM_Control_Module_1 = (1U << 1)`,  
`kPWM_Control_Module_2 = (1U << 2)`,  
`kPWM_Control_Module_3 = (1U << 3)` }  
*Options for submodule master control operation.*

## Functions

- void `PWM_SetupInputCapture` (`PWM_Type *base`, `pwm_submodule_t subModule`, `pwm_channels_t pwmChannel`, `const pwm_input_capture_param_t *inputCaptureParams`)  
*Sets up the PWM input capture.*
- void `PWM_SetupFaultInputFilter` (`PWM_Type *base`, `const pwm_fault_input_filter_param_t *faultInputFilterParams`)  
*Sets up the PWM fault input filter.*
- void `PWM_SetupFaults` (`PWM_Type *base`, `pwm_fault_input_t faultNum`, `const pwm_fault_param_t *faultParams`)  
*Sets up the PWM fault protection.*
- void `PWM_FaultDefaultConfig` (`pwm_fault_param_t *config`)  
*Fill in the PWM fault config struct with the default settings.*
- void `PWM_SetupForceSignal` (`PWM_Type *base`, `pwm_submodule_t subModule`, `pwm_channels_t pwmChannel`, `pwm_force_signal_t mode`)  
*Selects the signal to output on a PWM pin when a FORCE\_OUT signal is asserted.*
- static void `PWM_SetVALxValue` (`PWM_Type *base`, `pwm_submodule_t subModule`, `pwm_value_register_t valueRegister`, `uint16_t value`)  
*Set the PWM VALx registers.*
- static `uint16_t PWM_GetVALxValue` (`PWM_Type *base`, `pwm_submodule_t subModule`, `pwm_value_register_t valueRegister`)  
*Get the PWM VALx registers.*
- static void `PWM_OutputTriggerEnable` (`PWM_Type *base`, `pwm_submodule_t subModule`, `pwm_value_register_t valueRegister`, `bool activate`)



- Enables or disables the PWM output trigger.*

  - static void `PWM_ActivateOutputTrigger` (PWM\_Type \*base, `pwm_submodule_t` subModule, uint16\_t valueRegisterMask)
- Enables the PWM output trigger.*

  - static void `PWM_DeactivateOutputTrigger` (PWM\_Type \*base, `pwm_submodule_t` subModule, uint16\_t valueRegisterMask)
- Disables the PWM output trigger.*

  - static void `PWM_SetupSwCtrlOut` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_channels_t` pwmChannel, bool value)
- Sets the software control output for a pin to high or low.*

  - static void `PWM_SetPwmLdok` (PWM\_Type \*base, uint8\_t subModulesToUpdate, bool value)
- Sets or clears the PWM LDOK bit on a single or multiple submodules.*

  - static void `PWM_SetPwmFaultState` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_channels_t` pwmChannel, `pwm_fault_state_t` faultState)
- Set PWM output fault status.*

  - static void `PWM_SetupFaultDisableMap` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_channels_t` pwmChannel, `pwm_fault_channels_t` pwm\_fault\_channels, uint16\_t value)
- Set PWM fault disable mapping.*

  - static void `PWM_OutputEnable` (PWM\_Type \*base, `pwm_channels_t` pwmChannel, `pwm_submodule_t` subModule)
- Set PWM output enable.*

  - static void `PWM_OutputDisable` (PWM\_Type \*base, `pwm_channels_t` pwmChannel, `pwm_submodule_t` subModule)
- Set PWM output disable.*

  - uint8\_t `PWM_GetPwmChannelState` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_channels_t` pwmChannel)
- Get the dutycycle value.*

  - `status_t` `PWM_SetOutputToIdle` (PWM\_Type \*base, `pwm_channels_t` pwmChannel, `pwm_submodule_t` subModule, bool idleStatus)
- Set PWM output in idle status (high or low).*

  - void `PWM_SetClockMode` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_clock_prescale_t` prescaler)
- Set the pwm submodule prescaler.*

  - void `PWM_SetPwmForceOutputToZero` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_channels_t` pwmChannel, bool forcetozero)
- This function enables-disables the forcing of the output of a given eFlexPwm channel to logic 0.*

  - void `PWM_SetChannelOutput` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_channels_t` pwmChannel, `pwm_output_state_t` outputstate)
- This function set the output state of the PWM pin as requested for the current cycle.*

## Driver version

- #define `FSL_PWM_DRIVER_VERSION` (MAKE\_VERSION(2, 6, 1))  
Version 2.6.1.

## Initialization and deinitialization

- `status_t` `PWM_Init` (PWM\_Type \*base, `pwm_submodule_t` subModule, const `pwm_config_t` \*config)



- *Ungates the PWM submodule clock and configures the peripheral for basic operation.*
- void `PWM_Deinit` (PWM\_Type \*base, `pwm_submodule_t` subModule)  
*Gate the PWM submodule clock.*
- void `PWM_GetDefaultConfig` (`pwm_config_t` \*config)  
*Fill in the PWM config struct with the default settings.*

## Module PWM output

- `status_t PWM_SetupPwm` (PWM\_Type \*base, `pwm_submodule_t` subModule, const `pwm_signal_param_t` \*chnlParams, uint8\_t numOfChnls, `pwm_mode_t` mode, uint32\_t pwmFreq\_Hz, uint32\_t srcClock\_Hz)  
*Sets up the PWM signals for a PWM submodule.*
- `status_t PWM_SetupPwmPhaseShift` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_channels_t` pwmChannel, uint32\_t pwmFreq\_Hz, uint32\_t srcClock\_Hz, uint8\_t shiftvalue, bool doSync)  
*Set PWM phase shift for PWM channel running on channel PWM\_A, PWM\_B which with 50% duty cycle.*
- void `PWM_UpdatePwmDutycycle` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_channels_t` pwmSignal, `pwm_mode_t` currPwmMode, uint8\_t dutyCyclePercent)  
*Updates the PWM signal's dutycycle.*
- void `PWM_UpdatePwmDutycycleHighAccuracy` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_channels_t` pwmSignal, `pwm_mode_t` currPwmMode, uint16\_t dutyCycle)  
*Updates the PWM signal's dutycycle with 16-bit accuracy.*

## Interrupts Interface

- void `PWM_EnableInterrupts` (PWM\_Type \*base, `pwm_submodule_t` subModule, uint32\_t mask)  
*Enables the selected PWM interrupts.*
- void `PWM_DisableInterrupts` (PWM\_Type \*base, `pwm_submodule_t` subModule, uint32\_t mask)  
*Disables the selected PWM interrupts.*
- uint32\_t `PWM_GetEnabledInterrupts` (PWM\_Type \*base, `pwm_submodule_t` subModule)  
*Gets the enabled PWM interrupts.*

## DMA Interface

- static void `PWM_DMAFIFOWatermarkControl` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_watermark_control_t` pwm\_watermark\_control)  
*Capture DMA Enable Source Select.*
- static void `PWM_DMACaptureSourceSelect` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_dma_source_select_t` pwm\_dma\_source\_select)  
*Capture DMA Enable Source Select.*
- static void `PWM_EnabledDMACapture` (PWM\_Type \*base, `pwm_submodule_t` subModule, uint16\_t mask, bool activate)  
*Enables or disables the selected PWM DMA Capture read request.*
- static void `PWM_EnabledDMAWrite` (PWM\_Type \*base, `pwm_submodule_t` subModule, bool activate)  
*Enables or disables the PWM DMA write request.*

## Status Interface

- uint32\_t `PWM_GetStatusFlags` (PWM\_Type \*base, `pwm_submodule_t` subModule)

*Gets the PWM status flags.*

- void [PWM\\_ClearStatusFlags](#) (PWM\_Type \*base, [pwm\\_submodule\\_t](#) subModule, uint32\_t mask)  
*Clears the PWM status flags.*

## Timer Start and Stop

- static void [PWM\\_StartTimer](#) (PWM\_Type \*base, uint8\_t subModulesToStart)  
*Starts the PWM counter for a single or multiple submodules.*
- static void [PWM\\_StopTimer](#) (PWM\_Type \*base, uint8\_t subModulesToStop)  
*Stops the PWM counter for a single or multiple submodules.*

## 36.5 Data Structure Documentation

### 36.5.1 struct pwm\_signal\_param\_t

#### Data Fields

- [pwm\\_channels\\_t](#) pwmChannel  
*PWM channel being configured; PWM A or PWM B.*
- uint8\_t [dutyCyclePercent](#)  
*PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)...*
- [pwm\\_level\\_select\\_t](#) level  
*PWM output active level select.*
- uint16\_t [deadtimeValue](#)  
*The deadtime value; only used if channel pair is operating in complementary mode.*
- [pwm\\_fault\\_state\\_t](#) faultState  
*PWM output fault status.*
- bool [pwmchannelenable](#)  
*Enable PWM output.*

#### Field Documentation

##### (1) uint8\_t pwm\_signal\_param\_t::dutyCyclePercent

100=always active signal (100% duty cycle)

### 36.5.2 struct pwm\_config\_t

This structure holds the configuration settings for the PWM peripheral. To initialize this structure to reasonable defaults, call the [PWM\\_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 [enableDebugMode](#)

- *true: PWM continues to run in debug mode; false: PWM is paused in debug mode*
- bool `enableWait`  
*true: PWM continues to run in WAIT mode; false: PWM is paused in WAIT mode*
- `pwm_init_source_t` `initializationControl`  
*Option to initialize the counter.*
- `pwm_clock_source_t` `clockSource`  
*Clock source for the counter.*
- `pwm_clock_prescale_t` `prescale`  
*Pre-scaler to divide down the clock.*
- `pwm_chnl_pair_operation_t` `pairOperation`  
*Channel pair in independent or complementary mode.*
- `pwm_register_reload_t` `reloadLogic`  
*PWM Reload logic setup.*
- `pwm_reload_source_select_t` `reloadSelect`  
*Reload source select.*
- `pwm_load_frequency_t` `reloadFrequency`  
*Specifies when to reload, used when user's choice is not immediate reload.*
- `pwm_force_output_trigger_t` `forceTrigger`  
*Specify which signal will trigger a FORCE\_OUT.*

### 36.5.3 struct `pwm_fault_input_filter_param_t`

#### Data Fields

- `uint8_t` `faultFilterCount`  
*Fault filter count.*
- `uint8_t` `faultFilterPeriod`  
*Fault filter period; value of 0 will bypass the filter.*
- bool `faultGlitchStretch`  
*Fault Glitch Stretch Enable: A logic 1 means that input fault signals will be stretched to at least 2 IPBus clock cycles.*

### 36.5.4 struct `pwm_fault_param_t`

#### Data Fields

- `pwm_fault_clear_t` `faultClearingMode`  
*Fault clearing mode to use.*
- bool `faultLevel`  
*true: Logic 1 indicates fault; false: Logic 0 indicates fault*
- bool `enableCombinationalPath`  
*true: Combinational Path from fault input is enabled; false: No combination path is available*
- `pwm_fault_recovery_mode_t` `recoverMode`  
*Specify when to re-enable the PWM output.*

### 36.5.5 struct pwm\_input\_capture\_param\_t

#### Data Fields

- bool [captureInputSel](#)  
*true: Use the edge counter signal as source false: Use the raw input signal from the pin as source*
- uint8\_t [edgeCompareValue](#)  
*Compare value, used only if edge counter is used as source.*
- [pwm\\_input\\_capture\\_edge\\_t edge0](#)  
*Specify which edge causes a capture for input circuitry 0.*
- [pwm\\_input\\_capture\\_edge\\_t edge1](#)  
*Specify which edge causes a capture for input circuitry 1.*
- bool [enableOneShotCapture](#)  
*true: Use one-shot capture mode; false: Use free-running capture mode*
- uint8\_t [fifoWatermark](#)  
*Watermark level for capture FIFO.*

#### Field Documentation

##### (1) uint8\_t pwm\_input\_capture\_param\_t::fifoWatermark

The capture flags in the status register will set if the word count in the FIFO is greater than this watermark level

## 36.6 Macro Definition Documentation

### 36.6.1 #define PWM\_SUBMODULE\_CHANNEL 2

## 36.7 Enumeration Type Documentation

### 36.7.1 enum pwm\_submodule\_t

Enumerator

- kPWM\_Module\_0* Submodule 0.
- kPWM\_Module\_1* Submodule 1.
- kPWM\_Module\_2* Submodule 2.
- kPWM\_Module\_3* Submodule 3.

### 36.7.2 enum pwm\_value\_register\_t

Enumerator

- kPWM\_ValueRegister\_0* PWM Value0 register.
- kPWM\_ValueRegister\_1* PWM Value1 register.
- kPWM\_ValueRegister\_2* PWM Value2 register.
- kPWM\_ValueRegister\_3* PWM Value3 register.

*kPWM\_ValueRegister\_4* PWM Value4 register.

*kPWM\_ValueRegister\_5* PWM Value5 register.

### 36.7.3 enum \_pwm\_value\_register\_mask

Enumerator

*kPWM\_ValueRegisterMask\_0* PWM Value0 register mask.

*kPWM\_ValueRegisterMask\_1* PWM Value1 register mask.

*kPWM\_ValueRegisterMask\_2* PWM Value2 register mask.

*kPWM\_ValueRegisterMask\_3* PWM Value3 register mask.

*kPWM\_ValueRegisterMask\_4* PWM Value4 register mask.

*kPWM\_ValueRegisterMask\_5* PWM Value5 register mask.

### 36.7.4 enum pwm\_clock\_source\_t

Enumerator

*kPWM\_BusClock* The IPBus clock is used as the clock.

*kPWM\_ExternalClock* EXT\_CLK is used as the clock.

*kPWM\_Submodule0Clock* Clock of the submodule 0 (AUX\_CLK) is used as the source clock.

### 36.7.5 enum pwm\_clock\_prescale\_t

Enumerator

*kPWM\_Prescale\_Divide\_1* PWM clock frequency = fclk/1.

*kPWM\_Prescale\_Divide\_2* PWM clock frequency = fclk/2.

*kPWM\_Prescale\_Divide\_4* PWM clock frequency = fclk/4.

*kPWM\_Prescale\_Divide\_8* PWM clock frequency = fclk/8.

*kPWM\_Prescale\_Divide\_16* PWM clock frequency = fclk/16.

*kPWM\_Prescale\_Divide\_32* PWM clock frequency = fclk/32.

*kPWM\_Prescale\_Divide\_64* PWM clock frequency = fclk/64.

*kPWM\_Prescale\_Divide\_128* PWM clock frequency = fclk/128.

### 36.7.6 enum pwm\_force\_output\_trigger\_t

Enumerator

*kPWM\_Force\_Local* The local force signal, CTRL2[FORCE], from the submodule is used to force updates.

- kPWM\_Force\_Master* The master force signal from submodule 0 is used to force updates.
- kPWM\_Force\_LocalReload* The local reload signal from this submodule is used to force updates without regard to the state of LDOK.
- kPWM\_Force\_MasterReload* The master reload signal from submodule 0 is used to force updates if LDOK is set.
- kPWM\_Force\_LocalSync* The local sync signal from this submodule is used to force updates.
- kPWM\_Force\_MasterSync* The master sync signal from submodule 0 is used to force updates.
- kPWM\_Force\_External* The external force signal, EXT\_FORCE, from outside the PWM module causes updates.
- kPWM\_Force\_ExternalSync* The external sync signal, EXT\_SYNC, from outside the PWM module causes updates.

### 36.7.7 enum pwm\_output\_state\_t

Enumerator

- kPWM\_HighState* The output state of PWM channel is high.
- kPWM\_LowState* The output state of PWM channel is low.
- kPWM\_NormalState* The output state of PWM channel is normal.
- kPWM\_InvertState* The output state of PWM channel is invert.
- kPWM\_MaskState* The output state of PWM channel is mask.

### 36.7.8 enum pwm\_init\_source\_t

Enumerator

- kPWM\_Initialize\_LocalSync* Local sync causes initialization.
- kPWM\_Initialize\_MasterReload* Master reload from submodule 0 causes initialization.
- kPWM\_Initialize\_MasterSync* Master sync from submodule 0 causes initialization.
- kPWM\_Initialize\_ExtSync* EXT\_SYNC causes initialization.

### 36.7.9 enum pwm\_load\_frequency\_t

Enumerator

- kPWM\_LoadEveryOpportunity* Every PWM opportunity.
- kPWM\_LoadEvery2Opportunity* Every 2 PWM opportunities.
- kPWM\_LoadEvery3Opportunity* Every 3 PWM opportunities.
- kPWM\_LoadEvery4Opportunity* Every 4 PWM opportunities.
- kPWM\_LoadEvery5Opportunity* Every 5 PWM opportunities.
- kPWM\_LoadEvery6Opportunity* Every 6 PWM opportunities.

|                                    |                             |
|------------------------------------|-----------------------------|
| <i>kPWM_LoadEvery7Opportunity</i>  | Every 7 PWM opportunities.  |
| <i>kPWM_LoadEvery8Opportunity</i>  | Every 8 PWM opportunities.  |
| <i>kPWM_LoadEvery9Opportunity</i>  | Every 9 PWM opportunities.  |
| <i>kPWM_LoadEvery10Opportunity</i> | Every 10 PWM opportunities. |
| <i>kPWM_LoadEvery11Opportunity</i> | Every 11 PWM opportunities. |
| <i>kPWM_LoadEvery12Opportunity</i> | Every 12 PWM opportunities. |
| <i>kPWM_LoadEvery13Opportunity</i> | Every 13 PWM opportunities. |
| <i>kPWM_LoadEvery14Opportunity</i> | Every 14 PWM opportunities. |
| <i>kPWM_LoadEvery15Opportunity</i> | Every 15 PWM opportunities. |
| <i>kPWM_LoadEvery16Opportunity</i> | Every 16 PWM opportunities. |

### 36.7.10 enum pwm\_fault\_input\_t

Enumerator

|                     |                    |
|---------------------|--------------------|
| <i>kPWM_Fault_0</i> | Fault 0 input pin. |
| <i>kPWM_Fault_1</i> | Fault 1 input pin. |
| <i>kPWM_Fault_2</i> | Fault 2 input pin. |
| <i>kPWM_Fault_3</i> | Fault 3 input pin. |

### 36.7.11 enum pwm\_fault\_disable\_t

Enumerator

|                            |                          |
|----------------------------|--------------------------|
| <i>kPWM_FaultDisable_0</i> | Fault 0 disable mapping. |
| <i>kPWM_FaultDisable_1</i> | Fault 1 disable mapping. |
| <i>kPWM_FaultDisable_2</i> | Fault 2 disable mapping. |
| <i>kPWM_FaultDisable_3</i> | Fault 3 disable mapping. |

### 36.7.12 enum pwm\_input\_capture\_edge\_t

Enumerator

|                             |                                    |
|-----------------------------|------------------------------------|
| <i>kPWM_Disable</i>         | Disabled.                          |
| <i>kPWM_FallingEdge</i>     | Capture on falling edge only.      |
| <i>kPWM_RisingEdge</i>      | Capture on rising edge only.       |
| <i>kPWM_RiseAndFallEdge</i> | Capture on rising or falling edge. |

### 36.7.13 enum pwm\_force\_signal\_t

Enumerator

- kPWM\_UsePwm* Generated PWM signal is used by the deadtime logic.
- kPWM\_InvertedPwm* Inverted PWM signal is used by the deadtime logic.
- kPWM\_SoftwareControl* Software controlled value is used by the deadtime logic.
- kPWM\_UseExternal* PWM\_EXT\_A signal is used by the deadtime logic.

### 36.7.14 enum pwm\_chnl\_pair\_operation\_t

Enumerator

- kPWM\_Independent* PWM A & PWM B operate as 2 independent channels.
- kPWM\_ComplementaryPwmA* PWM A & PWM B are complementary channels, PWM A generates the signal.
- kPWM\_ComplementaryPwmB* PWM A & PWM B are complementary channels, PWM B generates the signal.

### 36.7.15 enum pwm\_register\_reload\_t

Enumerator

- kPWM\_ReloadImmediate* Buffered-registers get loaded with new values as soon as LDOK bit is set.
- kPWM\_ReloadPwmHalfCycle* Registers loaded on a PWM half cycle.
- kPWM\_ReloadPwmFullCycle* Registers loaded on a PWM full cycle.
- kPWM\_ReloadPwmHalfAndFullCycle* Registers loaded on a PWM half & full cycle.

### 36.7.16 enum pwm\_fault\_recovery\_mode\_t

Enumerator

- kPWM\_NoRecovery* PWM output will stay inactive.
- kPWM\_RecoverHalfCycle* PWM output re-enabled at the first half cycle.
- kPWM\_RecoverFullCycle* PWM output re-enabled at the first full cycle.
- kPWM\_RecoverHalfAndFullCycle* PWM output re-enabled at the first half or full cycle.



### 36.7.17 enum pwm\_interrupt\_enable\_t

Enumerator

|                                        |                             |
|----------------------------------------|-----------------------------|
| <i>kPWM_CompareVal0InterruptEnable</i> | PWM VAL0 compare interrupt. |
| <i>kPWM_CompareVal1InterruptEnable</i> | PWM VAL1 compare interrupt. |
| <i>kPWM_CompareVal2InterruptEnable</i> | PWM VAL2 compare interrupt. |
| <i>kPWM_CompareVal3InterruptEnable</i> | PWM VAL3 compare interrupt. |
| <i>kPWM_CompareVal4InterruptEnable</i> | PWM VAL4 compare interrupt. |
| <i>kPWM_CompareVal5InterruptEnable</i> | PWM VAL5 compare interrupt. |
| <i>kPWM_CaptureX0InterruptEnable</i>   | PWM capture X0 interrupt.   |
| <i>kPWM_CaptureX1InterruptEnable</i>   | PWM capture X1 interrupt.   |
| <i>kPWM_CaptureB0InterruptEnable</i>   | PWM capture B0 interrupt.   |
| <i>kPWM_CaptureB1InterruptEnable</i>   | PWM capture B1 interrupt.   |
| <i>kPWM_CaptureA0InterruptEnable</i>   | PWM capture A0 interrupt.   |
| <i>kPWM_CaptureA1InterruptEnable</i>   | PWM capture A1 interrupt.   |
| <i>kPWM_ReloadInterruptEnable</i>      | PWM reload interrupt.       |
| <i>kPWM_ReloadErrorInterruptEnable</i> | PWM reload error interrupt. |
| <i>kPWM_Fault0InterruptEnable</i>      | PWM fault 0 interrupt.      |
| <i>kPWM_Fault1InterruptEnable</i>      | PWM fault 1 interrupt.      |
| <i>kPWM_Fault2InterruptEnable</i>      | PWM fault 2 interrupt.      |
| <i>kPWM_Fault3InterruptEnable</i>      | PWM fault 3 interrupt.      |

### 36.7.18 enum pwm\_status\_flags\_t

Enumerator

|                             |                             |
|-----------------------------|-----------------------------|
| <i>kPWM_CompareVal0Flag</i> | PWM VAL0 compare flag.      |
| <i>kPWM_CompareVal1Flag</i> | PWM VAL1 compare flag.      |
| <i>kPWM_CompareVal2Flag</i> | PWM VAL2 compare flag.      |
| <i>kPWM_CompareVal3Flag</i> | PWM VAL3 compare flag.      |
| <i>kPWM_CompareVal4Flag</i> | PWM VAL4 compare flag.      |
| <i>kPWM_CompareVal5Flag</i> | PWM VAL5 compare flag.      |
| <i>kPWM_CaptureX0Flag</i>   | PWM capture X0 flag.        |
| <i>kPWM_CaptureX1Flag</i>   | PWM capture X1 flag.        |
| <i>kPWM_CaptureB0Flag</i>   | PWM capture B0 flag.        |
| <i>kPWM_CaptureB1Flag</i>   | PWM capture B1 flag.        |
| <i>kPWM_CaptureA0Flag</i>   | PWM capture A0 flag.        |
| <i>kPWM_CaptureA1Flag</i>   | PWM capture A1 flag.        |
| <i>kPWM_ReloadFlag</i>      | PWM reload flag.            |
| <i>kPWM_ReloadErrorFlag</i> | PWM reload error flag.      |
| <i>kPWM_RegUpdatedFlag</i>  | PWM registers updated flag. |
| <i>kPWM_Fault0Flag</i>      | PWM fault 0 flag.           |
| <i>kPWM_Fault1Flag</i>      | PWM fault 1 flag.           |

*kPWM\_Fault2Flag* PWM fault 2 flag.

*kPWM\_Fault3Flag* PWM fault 3 flag.

### 36.7.19 enum pwm\_dma\_enable\_t

Enumerator

*kPWM\_CaptureX0DMAEnable* PWM capture X0 DMA.

*kPWM\_CaptureX1DMAEnable* PWM capture X1 DMA.

*kPWM\_CaptureB0DMAEnable* PWM capture B0 DMA.

*kPWM\_CaptureB1DMAEnable* PWM capture B1 DMA.

*kPWM\_CaptureA0DMAEnable* PWM capture A0 DMA.

*kPWM\_CaptureA1DMAEnable* PWM capture A1 DMA.

### 36.7.20 enum pwm\_dma\_source\_select\_t

Enumerator

*kPWM\_DMAResourceDisable* Read DMA requests disabled.

*kPWM\_DMAWatermarksEnable* Exceeding a FIFO watermark sets the DMA read request.

*kPWM\_DMALocalSync* A local sync (VAL1 matches counter) sets the read DMA request.

*kPWM\_DMALocalReload* A local reload (STS[RF] being set) sets the read DMA request.

### 36.7.21 enum pwm\_watermark\_control\_t

Enumerator

*kPWM\_FIFOWatermarksOR* Selected FIFO watermarks are OR'ed together.

*kPWM\_FIFOWatermarksAND* Selected FIFO watermarks are AND'ed together.

### 36.7.22 enum pwm\_mode\_t

Enumerator

*kPWM\_SignedCenterAligned* Signed center-aligned.

*kPWM\_CenterAligned* Unsigned center-aligned.

*kPWM\_SignedEdgeAligned* Signed edge-aligned.

*kPWM\_EdgeAligned* Unsigned edge-aligned.

**36.7.23 enum pwm\_level\_select\_t**

Enumerator

*kPWM\_HighTrue* High level represents "on" or "active" state.*kPWM\_LowTrue* Low level represents "on" or "active" state.**36.7.24 enum pwm\_fault\_state\_t**

Enumerator

*kPWM\_PwmFaultState0* Output is forced to logic 0 state prior to consideration of output polarity control.*kPWM\_PwmFaultState1* Output is forced to logic 1 state prior to consideration of output polarity control.*kPWM\_PwmFaultState2* Output is tristated.*kPWM\_PwmFaultState3* Output is tristated.**36.7.25 enum pwm\_reload\_source\_select\_t**

Enumerator

*kPWM\_LocalReload* The local reload signal is used to reload registers.*kPWM\_MasterReload* The master reload signal (from submodule 0) is used to reload.**36.7.26 enum pwm\_fault\_clear\_t**

Enumerator

*kPWM\_Automatic* Automatic fault clearing.*kPWM\_ManualNormal* Manual fault clearing with no fault safety mode.*kPWM\_ManualSafety* Manual fault clearing with fault safety mode.**36.7.27 enum pwm\_module\_control\_t**

Enumerator

*kPWM\_Control\_Module\_0* Control submodule 0's start/stop,buffer reload operation.*kPWM\_Control\_Module\_1* Control submodule 1's start/stop,buffer reload operation.*kPWM\_Control\_Module\_2* Control submodule 2's start/stop,buffer reload operation.*kPWM\_Control\_Module\_3* Control submodule 3's start/stop,buffer reload operation.

## 36.8 Function Documentation

### 36.8.1 `status_t PWM_Init ( PWM_Type * base, pwm_submodule_t subModule, const pwm_config_t * config )`

Note

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

Parameters

|                  |                                         |
|------------------|-----------------------------------------|
| <i>base</i>      | PWM peripheral base address             |
| <i>subModule</i> | PWM submodule to configure              |
| <i>config</i>    | Pointer to user's PWM config structure. |

Returns

kStatus\_Success means success; else failed.

### 36.8.2 `void PWM_Deinit ( PWM_Type * base, pwm_submodule_t subModule )`

Parameters

|                  |                               |
|------------------|-------------------------------|
| <i>base</i>      | PWM peripheral base address   |
| <i>subModule</i> | PWM submodule to deinitialize |

### 36.8.3 `void PWM_GetDefaultConfig ( pwm_config_t * config )`

The default values are:

```
* config->enableDebugMode = false;
* config->enableWait = false;
* config->reloadSelect = kPWM_LocalReload;
* config->clockSource = kPWM_BusClock;
* config->prescale = kPWM_Prescale_Divide_1;
* config->initializationControl = kPWM_Initialize_LocalSync;
* config->forceTrigger = kPWM_Force_Local;
* config->reloadFrequency = kPWM_LoadEveryOpportunity;
* config->reloadLogic = kPWM_ReloadImmediate;
* config->pairOperation = kPWM_Independent;
*
```

## Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>config</i> | Pointer to user's PWM config structure. |
|---------------|-----------------------------------------|

**36.8.4** `status_t PWM_SetupPwm ( PWM_Type * base, pwm_submodule_t subModule, const pwm_signal_param_t * chnlParams, uint8_t numOfChnls, pwm_mode_t mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz )`

The function initializes the submodule according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements. If the dead time insertion logic is enabled, the pulse period is reduced by the dead time period specified by the user.

## Parameters

|                    |                                                                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | PWM peripheral base address                                                                                                                                      |
| <i>subModule</i>   | PWM submodule to configure                                                                                                                                       |
| <i>chnlParams</i>  | Array of PWM channel parameters to configure the channel(s), PWMX submodule is not supported.                                                                    |
| <i>numOfChnls</i>  | Number of channels to configure, this should be the size of the array passed in. Array size should not be more than 2 as each submodule has 2 pins to output PWM |
| <i>mode</i>        | PWM operation mode, options available in enumeration <a href="#">pwm_mode_t</a>                                                                                  |
| <i>pwmFreq_Hz</i>  | PWM signal frequency in Hz                                                                                                                                       |
| <i>srcClock_Hz</i> | PWM main counter clock in Hz.                                                                                                                                    |

## Returns

Returns `kStatus_Fail` if there was error setting up the signal; `kStatus_Success` otherwise

**36.8.5** `status_t PWM_SetupPwmPhaseShift ( PWM_Type * base, pwm_submodule_t subModule, pwm_channels_t pwmChannel, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, uint8_t shiftvalue, bool doSync )`

## Parameters

|                    |                                                                                                                   |
|--------------------|-------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | PWM peripheral base address                                                                                       |
| <i>subModule</i>   | PWM submodule to configure                                                                                        |
| <i>pwmChannel</i>  | PWM channel to configure                                                                                          |
| <i>pwmFreq_Hz</i>  | PWM signal frequency in Hz                                                                                        |
| <i>srcClock_Hz</i> | PWM main counter clock in Hz.                                                                                     |
| <i>shiftvalue</i>  | Phase shift value                                                                                                 |
| <i>doSync</i>      | true: Set LDOK bit for the submodule list; false: LDOK bit don't set, need to call PWM_SetPwmLdok to sync update. |

## Returns

Returns `kStatus_Fail` if there was error setting up the signal; `kStatus_Success` otherwise

### 36.8.6 void PWM\_UpdatePwmDutycycle ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_channels\_t *pwmSignal*, pwm\_mode\_t *currPwmMode*, uint8\_t *dutyCyclePercent* )

The function updates the PWM dutycyle to the new value that is passed in. If the dead time insertion logic is enabled then the pulse period is reduced by the dead time period specified by the user.

## Parameters

|                          |                                                                                                                               |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | PWM peripheral base address                                                                                                   |
| <i>subModule</i>         | PWM submodule to configure                                                                                                    |
| <i>pwmSignal</i>         | Signal (PWM A or PWM B) to update                                                                                             |
| <i>currPwmMode</i>       | The current PWM mode set during PWM setup                                                                                     |
| <i>dutyCycle-Percent</i> | New PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle) |

### 36.8.7 void PWM\_UpdatePwmDutycycleHighAccuracy ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_channels\_t *pwmSignal*, pwm\_mode\_t *currPwmMode*, uint16\_t *dutyCycle* )

The function updates the PWM dutycyle to the new value that is passed in. If the dead time insertion logic is enabled then the pulse period is reduced by the dead time period specified by the user.

## Parameters

|                    |                                                                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | PWM peripheral base address                                                                                                       |
| <i>subModule</i>   | PWM submodule to configure                                                                                                        |
| <i>pwmSignal</i>   | Signal (PWM A or PWM B) to update                                                                                                 |
| <i>currPwmMode</i> | The current PWM mode set during PWM setup                                                                                         |
| <i>dutyCycle</i>   | New PWM pulse width, value should be between 0 to 65535 0=inactive signal(0% duty cycle)... 65535=active signal (100% duty cycle) |

**36.8.8 void PWM\_SetupInputCapture ( PWM\_Type \* *base*, pwm\_- submodule\_t *subModule*, pwm\_channels\_t *pwmChannel*, const pwm\_input\_capture\_param\_t \* *inputCaptureParams* )**

Each PWM submodule has 3 pins that can be configured for use as input capture pins. This function sets up the capture parameters for each pin and enables the pin for input capture operation.

## Parameters

|                            |                                              |
|----------------------------|----------------------------------------------|
| <i>base</i>                | PWM peripheral base address                  |
| <i>subModule</i>           | PWM submodule to configure                   |
| <i>pwmChannel</i>          | Channel in the submodule to setup            |
| <i>inputCapture-Params</i> | Parameters passed in to set up the input pin |

**36.8.9 void PWM\_SetupFaultInputFilter ( PWM\_Type \* *base*, const pwm\_fault\_input\_filter\_param\_t \* *faultInputFilterParams* )**

## Parameters

|                                |                                                        |
|--------------------------------|--------------------------------------------------------|
| <i>base</i>                    | PWM peripheral base address                            |
| <i>faultInput-FilterParams</i> | Parameters passed in to set up the fault input filter. |

**36.8.10 void PWM\_SetupFaults ( PWM\_Type \* *base*, pwm\_fault\_input\_t *faultNum*, const pwm\_fault\_param\_t \* *faultParams* )**

PWM has 4 fault inputs.

## Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>base</i>        | PWM peripheral base address               |
| <i>faultNum</i>    | PWM fault to configure.                   |
| <i>faultParams</i> | Pointer to the PWM fault config structure |

**36.8.11 void PWM\_FaultDefaultConfig ( pwm\_fault\_param\_t \* config )**

The default values are:

```
* config->faultClearingMode = kPWM_Automatic;
* config->faultLevel = false;
* config->enableCombinationalPath = true;
* config->recoverMode = kPWM_NoRecovery;
*
```

## Parameters

|               |                                               |
|---------------|-----------------------------------------------|
| <i>config</i> | Pointer to user's PWM fault config structure. |
|---------------|-----------------------------------------------|

**36.8.12 void PWM\_SetupForceSignal ( PWM\_Type \* base, pwm\_submodule\_t subModule, pwm\_channels\_t pwmChannel, pwm\_force\_signal\_t mode )**

The user specifies which channel to configure by supplying the submodule number and whether to modify PWM A or PWM B within that submodule.

## Parameters

|                   |                                                |
|-------------------|------------------------------------------------|
| <i>base</i>       | PWM peripheral base address                    |
| <i>subModule</i>  | PWM submodule to configure                     |
| <i>pwmChannel</i> | Channel to configure                           |
| <i>mode</i>       | Signal to output when a FORCE_OUT is triggered |

**36.8.13 void PWM\_EnableInterrupts ( PWM\_Type \* base, pwm\_submodule\_t subModule, uint32\_t mask )**



## Parameters

|                  |                                                                                                                     |
|------------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | PWM peripheral base address                                                                                         |
| <i>subModule</i> | PWM submodule to configure                                                                                          |
| <i>mask</i>      | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">pwm_interrupt_enable_t</a> |

### 36.8.14 void PWM\_DisableInterrupts ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, uint32\_t *mask* )

## Parameters

|                  |                                                                                                                     |
|------------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | PWM peripheral base address                                                                                         |
| <i>subModule</i> | PWM submodule to configure                                                                                          |
| <i>mask</i>      | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">pwm_interrupt_enable_t</a> |

### 36.8.15 uint32\_t PWM\_GetEnabledInterrupts ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule* )

## Parameters

|                  |                             |
|------------------|-----------------------------|
| <i>base</i>      | PWM peripheral base address |
| <i>subModule</i> | PWM submodule to configure  |

## Returns

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

### 36.8.16 static void PWM\_DMAFIFOWatermarkControl ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_watermark\_control\_t *pwm\_watermark\_control* ) [inline], [static]

## Parameters

|                                          |                                |
|------------------------------------------|--------------------------------|
| <i>base</i>                              | PWM peripheral base address    |
| <i>subModule</i>                         | PWM submodule to configure     |
| <i>pwm_-<br/>watermark_-<br/>control</i> | PWM FIFO watermark and control |

**36.8.17** `static void PWM_DMACaptureSourceSelect ( PWM_Type * base, pwm_submodule_t subModule, pwm_dma_source_select_t pwm_dma_source_select ) [inline], [static]`

## Parameters

|                                    |                                      |
|------------------------------------|--------------------------------------|
| <i>base</i>                        | PWM peripheral base address          |
| <i>subModule</i>                   | PWM submodule to configure           |
| <i>pwm_dma_-<br/>source_select</i> | PWM capture DMA enable source select |

**36.8.18** `static void PWM_EnableDMACapture ( PWM_Type * base, pwm_submodule_t subModule, uint16_t mask, bool activate ) [inline], [static]`

## Parameters

|                  |                                                                                                                   |
|------------------|-------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | PWM peripheral base address                                                                                       |
| <i>subModule</i> | PWM submodule to configure                                                                                        |
| <i>mask</i>      | The DMA to enable or disable. This is a logical OR of members of the enumeration <a href="#">pwm_dma_enable_t</a> |
| <i>activate</i>  | true: Enable DMA read request; false: Disable DMA read request                                                    |

**36.8.19** `static void PWM_EnableDMAWrite ( PWM_Type * base, pwm_submodule_t subModule, bool activate ) [inline], [static]`

## Parameters

|                  |                                                                  |
|------------------|------------------------------------------------------------------|
| <i>base</i>      | PWM peripheral base address                                      |
| <i>subModule</i> | PWM submodule to configure                                       |
| <i>activate</i>  | true: Enable DMA write request; false: Disable DMA write request |

### 36.8.20 `uint32_t PWM_GetStatusFlags ( PWM_Type * base, pwm_submodule_t subModule )`

## Parameters

|                  |                             |
|------------------|-----------------------------|
| <i>base</i>      | PWM peripheral base address |
| <i>subModule</i> | PWM submodule to configure  |

## Returns

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

### 36.8.21 `void PWM_ClearStatusFlags ( PWM_Type * base, pwm_submodule_t subModule, uint32_t mask )`

## Parameters

|                  |                                                                                                                  |
|------------------|------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | PWM peripheral base address                                                                                      |
| <i>subModule</i> | PWM submodule to configure                                                                                       |
| <i>mask</i>      | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">pwm_status_flags_t</a> |

### 36.8.22 `static void PWM_StartTimer ( PWM_Type * base, uint8_t subModulesToStart ) [inline], [static]`

Sets the Run bit which enables the clocks to the PWM submodule. This function can start multiple submodules at the same time.

## Parameters

|                           |                                                                                                                   |
|---------------------------|-------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | PWM peripheral base address                                                                                       |
| <i>subModulesTo-Start</i> | PWM submodules to start. This is a logical OR of members of the enumeration <a href="#">pwm-_module_control_t</a> |

### 36.8.23 **static void PWM\_StopTimer ( PWM\_Type \* *base*, uint8\_t *subModulesToStop* ) [inline], [static]**

Clears the Run bit which resets the submodule's counter. This function can stop multiple submodules at the same time.

## Parameters

|                          |                                                                                                                  |
|--------------------------|------------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | PWM peripheral base address                                                                                      |
| <i>subModulesTo-Stop</i> | PWM submodules to stop. This is a logical OR of members of the enumeration <a href="#">pwm-_module_control_t</a> |

### 36.8.24 **static void PWM\_SetVALxValue ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_value\_register\_t *valueRegister*, uint16\_t *value* ) [inline], [static]**

This function allows the user to write value into VAL registers directly. And it will destroying the P-WM clock period set by the [PWM\\_SetupPwm\(\)/PWM\\_SetupPwmPhaseShift\(\)](#) functions. Due to VALx registers are buffered, the new value will not active unless call [PWM\\_SetPwmLdok\(\)](#) and the reload point is reached.

## Parameters

|                      |                                              |
|----------------------|----------------------------------------------|
| <i>base</i>          | PWM peripheral base address                  |
| <i>subModule</i>     | PWM submodule to configure                   |
| <i>valueRegister</i> | VALx register that will be written new value |
| <i>value</i>         | Value that will be write into VALx register  |

### 36.8.25 **static uint16\_t PWM\_GetVALxValue ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_value\_register\_t *valueRegister* ) [inline], [static]**

## Parameters

|                      |                                       |
|----------------------|---------------------------------------|
| <i>base</i>          | PWM peripheral base address           |
| <i>subModule</i>     | PWM submodule to configure            |
| <i>valueRegister</i> | VALx register that will be read value |

## Returns

The VALx register value

**36.8.26** `static void PWM_OutputTriggerEnable ( PWM_Type * base,  
pwm_submodule_t subModule, pwm_value_register_t valueRegister, bool  
activate ) [inline], [static]`

This function allows the user to enable or disable the PWM trigger. The PWM has 2 triggers. Trigger 0 is activated when the counter matches VAL 0, VAL 2, or VAL 4 register. Trigger 1 is activated when the counter matches VAL 1, VAL 3, or VAL 5 register.

## Parameters

|                      |                                                      |
|----------------------|------------------------------------------------------|
| <i>base</i>          | PWM peripheral base address                          |
| <i>subModule</i>     | PWM submodule to configure                           |
| <i>valueRegister</i> | Value register that will activate the trigger        |
| <i>activate</i>      | true: Enable the trigger; false: Disable the trigger |

**36.8.27** `static void PWM_ActivateOutputTrigger ( PWM_Type * base,  
pwm_submodule_t subModule, uint16_t valueRegisterMask ) [inline],  
[static]`

This function allows the user to enable one or more (VAL0-5) PWM trigger.

## Parameters

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

|                          |                                                                                                                          |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------|
| <i>subModule</i>         | PWM submodule to configure                                                                                               |
| <i>valueRegisterMask</i> | Value register mask that will activate one or more (VAL0-5) trigger enumeration <a href="#">_pwm_value_register_mask</a> |

**36.8.28** `static void PWM_DeactivateOutputTrigger ( PWM_Type * base, pwm_submodule_t subModule, uint16_t valueRegisterMask ) [inline], [static]`

This function allows the user to disables one or more (VAL0-5) PWM trigger.

Parameters

|                          |                                                                                                                            |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | PWM peripheral base address                                                                                                |
| <i>subModule</i>         | PWM submodule to configure                                                                                                 |
| <i>valueRegisterMask</i> | Value register mask that will Deactivate one or more (VAL0-5) trigger enumeration <a href="#">_pwm_value_register_mask</a> |

**36.8.29** `static void PWM_SetupSwCtrlOut ( PWM_Type * base, pwm_submodule_t subModule, pwm_channels_t pwmChannel, bool value ) [inline], [static]`

The user specifies which channel to modify by supplying the submodule number and whether to modify PWM A or PWM B within that submodule.

Parameters

|                   |                                                  |
|-------------------|--------------------------------------------------|
| <i>base</i>       | PWM peripheral base address                      |
| <i>subModule</i>  | PWM submodule to configure                       |
| <i>pwmChannel</i> | Channel to configure                             |
| <i>value</i>      | true: Supply a logic 1, false: Supply a logic 0. |

**36.8.30** `static void PWM_SetPwmLdok ( PWM_Type * base, uint8_t subModulesToUpdate, bool value ) [inline], [static]`

Set LDOK bit to load buffered values into CTRL[PRSC] and the INIT, FRACVAL and VAL registers. The values are loaded immediately if kPWM\_ReloadImmediate option was chosen during config. Else the values are loaded at the next PWM reload point. This function can issue the load command to multiple submodules at the same time.

## Parameters

|                           |                                                                                                                                        |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | PWM peripheral base address                                                                                                            |
| <i>subModulesToUpdate</i> | PWM submodules to update with buffered values. This is a logical OR of members of the enumeration <a href="#">pwm_module_control_t</a> |
| <i>value</i>              | true: Set LDOK bit for the submodule list; false: Clear LDOK bit                                                                       |

**36.8.31 static void PWM\_SetPwmFaultState ( PWM\_Type \* *base*,  
pwm\_submodule\_t *subModule*, pwm\_channels\_t *pwmChannel*,  
pwm\_fault\_state\_t *faultState* ) [inline], [static]**

These bits determine the fault state for the PWM\_A output in fault conditions and STOP mode. It may also define the output state in WAIT and DEBUG modes depending on the settings of CTRL2[WAITEN] and CTRL2[DBGEN]. This function can update PWM output fault status.

## Parameters

|                   |                             |
|-------------------|-----------------------------|
| <i>base</i>       | PWM peripheral base address |
| <i>subModule</i>  | PWM submodule to configure  |
| <i>pwmChannel</i> | Channel to configure        |
| <i>faultState</i> | PWM output fault status     |

**36.8.32 static void PWM\_SetupFaultDisableMap ( PWM\_Type \* *base*,  
pwm\_submodule\_t *subModule*, pwm\_channels\_t *pwmChannel*,  
pwm\_fault\_channels\_t *pwm\_fault\_channels*, uint16\_t *value* ) [inline],  
[static]**

Each of the four bits of this read/write field is one-to-one associated with the four FAULTx inputs of fault channel 0/1. The PWM output will be turned off if there is a logic 1 on an FAULTx input and a 1 in the corresponding bit of this field. A reset sets all bits in this field.

## Parameters

|                           |                                                                                  |
|---------------------------|----------------------------------------------------------------------------------|
| <i>base</i>               | PWM peripheral base address                                                      |
| <i>subModule</i>          | PWM submodule to configure                                                       |
| <i>pwmChannel</i>         | PWM channel to configure                                                         |
| <i>pwm_fault_channels</i> | PWM fault channel to configure                                                   |
| <i>value</i>              | Fault disable mapping mask value enumeration <a href="#">pwm_fault_disable_t</a> |

**36.8.33** `static void PWM_OutputEnable ( PWM_Type * base, pwm_channels_t pwmChannel, pwm_submodule_t subModule ) [inline], [static]`

This feature allows the user to enable the PWM Output.

Parameters

|                   |                             |
|-------------------|-----------------------------|
| <i>base</i>       | PWM peripheral base address |
| <i>pwmChannel</i> | PWM channel to configure    |
| <i>subModule</i>  | PWM submodule to configure  |

**36.8.34** `static void PWM_OutputDisable ( PWM_Type * base, pwm_channels_t pwmChannel, pwm_submodule_t subModule ) [inline], [static]`

This feature allows the user to disable the PWM output.

Parameters

|                   |                             |
|-------------------|-----------------------------|
| <i>base</i>       | PWM peripheral base address |
| <i>pwmChannel</i> | PWM channel to configure    |
| <i>subModule</i>  | PWM submodule to configure  |

**36.8.35** `uint8_t PWM_GetPwmChannelState ( PWM_Type * base, pwm_submodule_t subModule, pwm_channels_t pwmChannel )`

Parameters

|                   |                             |
|-------------------|-----------------------------|
| <i>base</i>       | PWM peripheral base address |
| <i>subModule</i>  | PWM submodule to configure  |
| <i>pwmChannel</i> | PWM channel to configure    |

Returns

Current channel dutycycle value.



### 36.8.36 `status_t PWM_SetOutputToldle ( PWM_Type * base, pwm_channels_t pwmChannel, pwm_submodule_t subModule, bool idleStatus )`

Note

This API should call after [PWM\\_SetupPwm\(\)](#) APIs, and PWMX submodule is not supported.

Parameters

|                   |                                                                                   |
|-------------------|-----------------------------------------------------------------------------------|
| <i>base</i>       | PWM peripheral base address                                                       |
| <i>pwmChannel</i> | PWM channel to configure                                                          |
| <i>subModule</i>  | PWM submodule to configure                                                        |
| <i>idleStatus</i> | True: PWM output is high in idle status; false: PWM output is low in idle status. |

Returns

kStatus\_Fail if there was error setting up the signal; kStatus\_Success if set output idle success

### 36.8.37 `void PWM_SetClockMode ( PWM_Type * base, pwm_submodule_t subModule, pwm_clock_prescale_t prescaler )`

Parameters

|                  |                             |
|------------------|-----------------------------|
| <i>base</i>      | PWM peripheral base address |
| <i>subModule</i> | PWM submodule to configure  |
| <i>prescaler</i> | Set prescaler value         |

### 36.8.38 `void PWM_SetPwmForceOutputToZero ( PWM_Type * base, pwm_submodule_t subModule, pwm_channels_t pwmChannel, bool forcetozero )`

Parameters

|                    |                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------|
| <i>base</i>        | PWM peripheral base address                                                                       |
| <i>pwmChannel</i>  | PWM channel to configure                                                                          |
| <i>subModule</i>   | PWM submodule to configure                                                                        |
| <i>forcetozero</i> | True: Enable the pwm force output to zero; False: Disable the pwm output resumes normal function. |

**36.8.39** void PWM\_SetChannelOutput ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_channels\_t *pwmChannel*, pwm\_output\_state\_t *outputstate* )

## Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>base</i>        | PWM peripheral base address                                    |
| <i>subModule</i>   | PWM submodule to configure                                     |
| <i>pwmChannel</i>  | PWM channel to configure                                       |
| <i>outputstate</i> | Set pwm output state, see <a href="#">pwm_output_state_t</a> . |

# Chapter 37

## QTMR: Quad Timer Driver

### 37.1 Overview

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

### Data Structures

- struct `qtmr_config_t`  
*Quad Timer config structure. [More...](#)*

### Enumerations

- enum `qtmr_primary_count_source_t` {  
    `kQTMR_ClockCounter0InputPin` = 0,  
    `kQTMR_ClockCounter1InputPin`,  
    `kQTMR_ClockCounter2InputPin`,  
    `kQTMR_ClockCounter3InputPin`,  
    `kQTMR_ClockCounter0Output`,  
    `kQTMR_ClockCounter1Output`,  
    `kQTMR_ClockCounter2Output`,  
    `kQTMR_ClockCounter3Output`,  
    `kQTMR_ClockDivide_1`,  
    `kQTMR_ClockDivide_2`,  
    `kQTMR_ClockDivide_4`,  
    `kQTMR_ClockDivide_8`,  
    `kQTMR_ClockDivide_16`,  
    `kQTMR_ClockDivide_32`,  
    `kQTMR_ClockDivide_64`,  
    `kQTMR_ClockDivide_128` }  
*Quad Timer primary clock source selection.*
- enum `qtmr_input_source_t` {  
    `kQTMR_Counter0InputPin` = 0,  
    `kQTMR_Counter1InputPin`,  
    `kQTMR_Counter2InputPin`,  
    `kQTMR_Counter3InputPin` }  
*Quad Timer input sources selection.*
- enum `qtmr_counting_mode_t` {

```

kQTMR_NoOperation = 0,
kQTMR_PriSrcRiseEdge,
kQTMR_PriSrcRiseAndFallEdge,
kQTMR_PriSrcRiseEdgeSecInpHigh,
kQTMR_QuadCountMode,
kQTMR_PriSrcRiseEdgeSecDir,
kQTMR_SecSrcTrigPriCnt,
kQTMR_CascadeCount }

```

*Quad Timer counting mode selection.*

- enum qtmr\_pwm\_out\_state\_t {

```

kQTMR_PwmLow = 0,
kQTMR_PwmHigh }

```

*Quad Timer PWM output state.*

- enum qtmr\_output\_mode\_t {

```

kQTMR_AssertWhenCountActive = 0,
kQTMR_ClearOnCompare,
kQTMR_SetOnCompare,
kQTMR_ToggleOnCompare,
kQTMR_ToggleOnAltCompareReg,
kQTMR_SetOnCompareClearOnSecSrcInp,
kQTMR_SetOnCompareClearOnCountRoll,
kQTMR_EnableGateClock }

```

*Quad Timer output mode selection.*

- enum qtmr\_input\_capture\_edge\_t {

```

kQTMR_NoCapture = 0,
kQTMR_RisingEdge,
kQTMR_FallingEdge,
kQTMR_RisingAndFallingEdge }

```

*Quad Timer input capture edge mode, rising edge, or falling edge.*

- enum qtmr\_preload\_control\_t {

```

kQTMR_NoPreload = 0,
kQTMR_LoadOnComp1,
kQTMR_LoadOnComp2 }

```

*Quad Timer input capture edge mode, rising edge, or falling edge.*

- enum qtmr\_debug\_action\_t {

```

kQTMR_RunNormalInDebug = 0U,
kQTMR_HaltCounter,
kQTMR_ForceOutToZero,
kQTMR_HaltCountForceOutZero }

```

*List of Quad Timer run options when in Debug mode.*

- enum qtmr\_interrupt\_enable\_t {

```

kQTMR_CompareInterruptEnable = (1U << 0),
kQTMR_Compare1InterruptEnable = (1U << 1),
kQTMR_Compare2InterruptEnable = (1U << 2),
kQTMR_OverflowInterruptEnable = (1U << 3),
kQTMR_EdgeInterruptEnable = (1U << 4) }

```

*List of Quad Timer interrupts.*

- enum `qtmr_status_flags_t` {  
`kQTMR_CompareFlag` = (1U << 0),  
`kQTMR_Compare1Flag` = (1U << 1),  
`kQTMR_Compare2Flag` = (1U << 2),  
`kQTMR_OverflowFlag` = (1U << 3),  
`kQTMR_EdgeFlag` = (1U << 4) }

*List of Quad Timer flags.*

- enum `qtmr_channel_selection_t` {  
`kQTMR_Channel_0` = 0U,  
`kQTMR_Channel_1`,  
`kQTMR_Channel_2`,  
`kQTMR_Channel_3` }

*List of channel selection.*

- enum `qtmr_dma_enable_t` {  
`kQTMR_InputEdgeFlagDmaEnable` = (1U << 0),  
`kQTMR_ComparatorPreload1DmaEnable` = (1U << 1),  
`kQTMR_ComparatorPreload2DmaEnable` = (1U << 2) }

*List of Quad Timer DMA enable.*

## Functions

- `status_t QTMR_SetupPwm` (TMR\_Type \*base, `qtmr_channel_selection_t` channel, uint32\_t pwmFreqHz, uint8\_t dutyCyclePercent, bool outputPolarity, uint32\_t srcClock\_Hz)

*Sets up Quad timer module for PWM signal output.*

- void `QTMR_SetupInputCapture` (TMR\_Type \*base, `qtmr_channel_selection_t` channel, `qtmr_input_source_t` capturePin, bool inputPolarity, bool reloadOnCapture, `qtmr_input_capture_edge_t` captureMode)

*Allows the user to count the source clock cycles until a capture event arrives.*

## Driver version

- #define `FSL_QTMR_DRIVER_VERSION` (MAKE\_VERSION(2, 2, 2))  
*Version.*

## Initialization and deinitialization

- void `QTMR_Init` (TMR\_Type \*base, `qtmr_channel_selection_t` channel, const `qtmr_config_t` \*config)

*Ungates the Quad Timer clock and configures the peripheral for basic operation.*

- void `QTMR_Deinit` (TMR\_Type \*base, `qtmr_channel_selection_t` channel)

*Stops the counter and gates the Quad Timer clock.*

- void `QTMR_GetDefaultConfig` (`qtmr_config_t` \*config)

*Fill in the Quad Timer config struct with the default settings.*

## Interrupt Interface

- void `QTMR_EnableInterrupts` (TMR\_Type \*base, `qtmr_channel_selection_t` channel, uint32\_t mask)

- *Enables the selected Quad Timer interrupts.*  
void [QTMR\\_DisableInterrupts](#) (TMR\_Type \*base, [qtmr\\_channel\\_selection\\_t](#) channel, uint32\_t mask)
- *Disables the selected Quad Timer interrupts.*  
uint32\_t [QTMR\\_GetEnabledInterrupts](#) (TMR\_Type \*base, [qtmr\\_channel\\_selection\\_t](#) channel)  
*Gets the enabled Quad Timer interrupts.*

## Status Interface

- uint32\_t [QTMR\\_GetStatus](#) (TMR\_Type \*base, [qtmr\\_channel\\_selection\\_t](#) channel)  
*Gets the Quad Timer status flags.*
- void [QTMR\\_ClearStatusFlags](#) (TMR\_Type \*base, [qtmr\\_channel\\_selection\\_t](#) channel, uint32\_t mask)  
*Clears the Quad Timer status flags.*

## Read and Write the timer period

- void [QTMR\\_SetTimerPeriod](#) (TMR\_Type \*base, [qtmr\\_channel\\_selection\\_t](#) channel, uint16\_t ticks)  
*Sets the timer period in ticks.*
- void [QTMR\\_SetCompareValue](#) (TMR\_Type \*base, [qtmr\\_channel\\_selection\\_t](#) channel, uint16\_t ticks)  
*Set compare value.*
- static void [QTMR\\_SetLoadValue](#) (TMR\_Type \*base, [qtmr\\_channel\\_selection\\_t](#) channel, uint16\_t value)  
*Set load value.*
- static uint16\_t [QTMR\\_GetCurrentTimerCount](#) (TMR\_Type \*base, [qtmr\\_channel\\_selection\\_t](#) channel)  
*Reads the current timer counting value.*

## Timer Start and Stop

- static void [QTMR\\_StartTimer](#) (TMR\_Type \*base, [qtmr\\_channel\\_selection\\_t](#) channel, [qtmr\\_counting\\_mode\\_t](#) clockSource)  
*Starts the Quad Timer counter.*
- static void [QTMR\\_StopTimer](#) (TMR\_Type \*base, [qtmr\\_channel\\_selection\\_t](#) channel)  
*Stops the Quad Timer counter.*

## Enable and Disable the Quad Timer DMA

- void [QTMR\\_EnableDma](#) (TMR\_Type \*base, [qtmr\\_channel\\_selection\\_t](#) channel, uint32\_t mask)  
*Enable the Quad Timer DMA.*
- void [QTMR\\_DisableDma](#) (TMR\_Type \*base, [qtmr\\_channel\\_selection\\_t](#) channel, uint32\_t mask)  
*Disable the Quad Timer DMA.*
- void [QTMR\\_SetPwmOutputToIdle](#) (TMR\_Type \*base, [qtmr\\_channel\\_selection\\_t](#) channel, bool idleStatus)  
*Set PWM output in idle status (high or low).*
- static [qtmr\\_pwm\\_out\\_state\\_t](#) [QTMR\\_GetPwmOutputStatus](#) (TMR\_Type \*base, [qtmr\\_channel\\_selection\\_t](#) channel)  
*Get the channel output status.*

- `uint8_t QTMR_GetPwmChannelStatus` (TMR\_Type \*base, `qtmr_channel_selection_t` channel)  
*Get the PWM channel dutycycle value.*
- `void QTMR_SetPwmClockMode` (TMR\_Type \*base, `qtmr_channel_selection_t` channel, `qtmr_primary_count_source_t` prescaler)  
*This function set the value of the prescaler on QTimer channels.*

## 37.2 Data Structure Documentation

### 37.2.1 struct qtmr\_config\_t

This structure holds the configuration settings for the Quad Timer peripheral. To initialize this structure to reasonable defaults, call the `QTMR_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

- `qtmr_primary_count_source_t primarySource`  
*Specify the primary count source.*
- `qtmr_input_source_t secondarySource`  
*Specify the secondary count source.*
- `bool enableMasterMode`  
*true: Broadcast compare function output to other counters; false no broadcast*
- `bool enableExternalForce`  
*true: Compare from another counter force state of OFLAG signal false: OFLAG controlled by local counter*
- `uint8_t faultFilterCount`  
*Fault filter count.*
- `uint8_t faultFilterPeriod`  
*Fault filter period; value of 0 will bypass the filter.*
- `qtmr_debug_action_t debugMode`  
*Operation in Debug mode.*

## 37.3 Enumeration Type Documentation

### 37.3.1 enum qtmr\_primary\_count\_source\_t

Enumerator

- `kQTMR_ClockCounter0InputPin` Use counter 0 input pin.
- `kQTMR_ClockCounter1InputPin` Use counter 1 input pin.
- `kQTMR_ClockCounter2InputPin` Use counter 2 input pin.
- `kQTMR_ClockCounter3InputPin` Use counter 3 input pin.
- `kQTMR_ClockCounter0Output` Use counter 0 output.
- `kQTMR_ClockCounter1Output` Use counter 1 output.
- `kQTMR_ClockCounter2Output` Use counter 2 output.
- `kQTMR_ClockCounter3Output` Use counter 3 output.



*kQTMR\_ClockDivide\_1* IP bus clock divide by 1 prescaler.  
*kQTMR\_ClockDivide\_2* IP bus clock divide by 2 prescaler.  
*kQTMR\_ClockDivide\_4* IP bus clock divide by 4 prescaler.  
*kQTMR\_ClockDivide\_8* IP bus clock divide by 8 prescaler.  
*kQTMR\_ClockDivide\_16* IP bus clock divide by 16 prescaler.  
*kQTMR\_ClockDivide\_32* IP bus clock divide by 32 prescaler.  
*kQTMR\_ClockDivide\_64* IP bus clock divide by 64 prescaler.  
*kQTMR\_ClockDivide\_128* IP bus clock divide by 128 prescaler.

### 37.3.2 enum qtmr\_input\_source\_t

Enumerator

*kQTMR\_Counter0InputPin* Use counter 0 input pin.  
*kQTMR\_Counter1InputPin* Use counter 1 input pin.  
*kQTMR\_Counter2InputPin* Use counter 2 input pin.  
*kQTMR\_Counter3InputPin* Use counter 3 input pin.

### 37.3.3 enum qtmr\_counting\_mode\_t

Enumerator

*kQTMR\_NoOperation* No operation.  
*kQTMR\_PriSrcRiseEdge* Count rising edges of primary source.  
*kQTMR\_PriSrcRiseAndFallEdge* Count rising and falling edges of primary source.  
*kQTMR\_PriSrcRiseEdgeSecInpHigh* Count rise edges of pri SRC while sec inp high active.  
*kQTMR\_QuadCountMode* Quadrature count mode, uses pri and sec sources.  
*kQTMR\_PriSrcRiseEdgeSecDir* Count rising edges of pri SRC; sec SRC specifies dir.  
*kQTMR\_SecSrcTrigPriCnt* Edge of sec SRC trigger primary count until compare.  
*kQTMR\_CascadeCount* Cascaded count mode (up/down)

### 37.3.4 enum qtmr\_pwm\_out\_state\_t

Enumerator

*kQTMR\_PwmLow* The output state of PWM channel is low.  
*kQTMR\_PwmHigh* The output state of PWM channel is high.

### 37.3.5 enum qtmr\_output\_mode\_t

Enumerator

*kQTMR\_AssertWhenCountActive* Assert OFLAG while counter is active.  
*kQTMR\_ClearOnCompare* Clear OFLAG on successful compare.  
*kQTMR\_SetOnCompare* Set OFLAG on successful compare.  
*kQTMR\_ToggleOnCompare* Toggle OFLAG on successful compare.  
*kQTMR\_ToggleOnAltCompareReg* Toggle OFLAG using alternating compare registers.  
*kQTMR\_SetOnCompareClearOnSecSrcInp* Set OFLAG on compare, clear on sec SRC input edge.  
  
*kQTMR\_SetOnCompareClearOnCountRoll* Set OFLAG on compare, clear on counter rollover.  
*kQTMR\_EnableGateClock* Enable gated clock output while count is active.

### 37.3.6 enum qtmr\_input\_capture\_edge\_t

Enumerator

*kQTMR\_NoCapture* Capture is disabled.  
*kQTMR\_RisingEdge* Capture on rising edge (IPS=0) or falling edge (IPS=1)  
*kQTMR\_FallingEdge* Capture on falling edge (IPS=0) or rising edge (IPS=1)  
*kQTMR\_RisingAndFallingEdge* Capture on both edges.

### 37.3.7 enum qtmr\_preload\_control\_t

Enumerator

*kQTMR\_NoPreload* Never preload.  
*kQTMR\_LoadOnComp1* Load upon successful compare with value in COMP1.  
*kQTMR\_LoadOnComp2* Load upon successful compare with value in COMP2.

### 37.3.8 enum qtmr\_debug\_action\_t

Enumerator

*kQTMR\_RunNormalInDebug* Continue with normal operation.  
*kQTMR\_HaltCounter* Halt counter.  
*kQTMR\_ForceOutToZero* Force output to logic 0.  
*kQTMR\_HaltCountForceOutZero* Halt counter and force output to logic 0.

### 37.3.9 enum qtmr\_interrupt\_enable\_t

Enumerator

*kQTMR\_CompareInterruptEnable* Compare interrupt.  
*kQTMR\_Compare1InterruptEnable* Compare 1 interrupt.  
*kQTMR\_Compare2InterruptEnable* Compare 2 interrupt.  
*kQTMR\_OverflowInterruptEnable* Timer overflow interrupt.  
*kQTMR\_EdgeInterruptEnable* Input edge interrupt.

### 37.3.10 enum qtmr\_status\_flags\_t

Enumerator

*kQTMR\_CompareFlag* Compare flag.  
*kQTMR\_Compare1Flag* Compare 1 flag.  
*kQTMR\_Compare2Flag* Compare 2 flag.  
*kQTMR\_OverflowFlag* Timer overflow flag.  
*kQTMR\_EdgeFlag* Input edge flag.

### 37.3.11 enum qtmr\_channel\_selection\_t

Enumerator

*kQTMR\_Channel\_0* TMR Channel 0.  
*kQTMR\_Channel\_1* TMR Channel 1.  
*kQTMR\_Channel\_2* TMR Channel 2.  
*kQTMR\_Channel\_3* TMR Channel 3.

### 37.3.12 enum qtmr\_dma\_enable\_t

Enumerator

*kQTMR\_InputEdgeFlagDmaEnable* Input Edge Flag DMA Enable.  
*kQTMR\_ComparatorPreload1DmaEnable* Comparator Preload Register 1 DMA Enable.  
*kQTMR\_ComparatorPreload2DmaEnable* Comparator Preload Register 2 DMA Enable.

## 37.4 Function Documentation

**37.4.1 void QTMR\_Init ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, const qtmr\_config\_t \* *config* )**

## Note

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

## Parameters

|                |                                               |
|----------------|-----------------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address            |
| <i>channel</i> | Quad Timer channel number                     |
| <i>config</i>  | Pointer to user's Quad Timer config structure |

### 37.4.2 void QTMR\_Deinit ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel* )

## Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |

### 37.4.3 void QTMR\_GetDefaultConfig ( qtmr\_config\_t \* *config* )

The default values are:

```
* config->debugMode = kQTMR_RunNormalInDebug;
* config->enableExternalForce = false;
* config->enableMasterMode = false;
* config->faultFilterCount = 0;
* config->faultFilterPeriod = 0;
* config->primarySource = kQTMR_ClockDivide_2;
* config->secondarySource = kQTMR_Counter0InputPin;
*
```

## Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>config</i> | Pointer to user's Quad Timer config structure. |
|---------------|------------------------------------------------|

### 37.4.4 status\_t QTMR\_SetupPwm ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, uint32\_t *pwmFreqHz*, uint8\_t *dutyCyclePercent*, bool *outputPolarity*, uint32\_t *srcClock\_Hz* )

The function initializes the timer module according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements.

## Parameters

|                          |                                                                                                                           |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | Quad Timer peripheral base address                                                                                        |
| <i>channel</i>           | Quad Timer channel number                                                                                                 |
| <i>pwmFreqHz</i>         | PWM signal frequency in Hz                                                                                                |
| <i>dutyCycle-Percent</i> | PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle) |
| <i>outputPolarity</i>    | true: invert polarity of the output signal, false: no inversion                                                           |
| <i>srcClock_Hz</i>       | Main counter clock in Hz.                                                                                                 |

## Returns

Returns an error if there was error setting up the signal.

**37.4.5 void QTMR\_SetupInputCapture ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, qtmr\_input\_source\_t *capturePin*, bool *inputPolarity*, bool *reloadOnCapture*, qtmr\_input\_capture\_edge\_t *captureMode* )**

The count is stored in the capture register.

## Parameters

|                         |                                                                         |
|-------------------------|-------------------------------------------------------------------------|
| <i>base</i>             | Quad Timer peripheral base address                                      |
| <i>channel</i>          | Quad Timer channel number                                               |
| <i>capturePin</i>       | Pin through which we receive the input signal to trigger the capture    |
| <i>inputPolarity</i>    | true: invert polarity of the input signal, false: no inversion          |
| <i>reloadOn-Capture</i> | true: reload the counter when an input capture occurs, false: no reload |
| <i>captureMode</i>      | Specifies which edge of the input signal triggers a capture             |

**37.4.6 void QTMR\_EnableInterrupts ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, uint32\_t *mask* )**

## Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address                                                                                   |
| <i>channel</i> | Quad Timer channel number                                                                                            |
| <i>mask</i>    | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">qtmr_interrupt_enable_t</a> |

### 37.4.7 void QTMR\_DisableInterrupts ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, uint32\_t *mask* )

## Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address                                                                                   |
| <i>channel</i> | Quad Timer channel number                                                                                            |
| <i>mask</i>    | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">qtmr_interrupt_enable_t</a> |

### 37.4.8 uint32\_t QTMR\_GetEnabledInterrupts ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel* )

## Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |

## Returns

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

### 37.4.9 uint32\_t QTMR\_GetStatus ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel* )

## Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |

## Returns

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

### 37.4.10 void QTMR\_ClearStatusFlags ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, uint32\_t *mask* )

## Parameters

|                |                                                                                                                   |
|----------------|-------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address                                                                                |
| <i>channel</i> | Quad Timer channel number                                                                                         |
| <i>mask</i>    | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">qtmr_status_flags_t</a> |

### 37.4.11 void QTMR\_SetTimerPeriod ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, uint16\_t *ticks* )

Timers counts from initial value till it equals the count value set here. The counter will then reinitialize to the value specified in the Load register.

## Note

1. This function will write the time period in ticks to COMP1 or COMP2 register depending on the count direction
2. User can call the utility macros provided in `fsl_common.h` to convert to ticks
3. This function supports cases, providing only primary source clock without secondary source clock.

## Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |
| <i>ticks</i>   | Timer period in units of ticks     |

### 37.4.12 void QTMR\_SetCompareValue ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, uint16\_t *ticks* )

This function sets the value used for comparison with the counter value.

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |
| <i>ticks</i>   | Timer period in units of ticks.    |

### 37.4.13 static void QTMR\_SetLoadValue ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, uint16\_t *value* ) [inline], [static]

This function sets the value used to initialize the counter after a counter comparison.

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address  |
| <i>channel</i> | Quad Timer channel number           |
| <i>value</i>   | Load register initialization value. |

### 37.4.14 static uint16\_t QTMR\_GetCurrentTimerCount ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel* ) [inline], [static]

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note

User can call the utility macros provided in `fsl_common.h` to convert ticks to usec or msec



## Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |

## Returns

Current counter value in ticks

**37.4.15** `static void QTMR_StartTimer ( TMR_Type * base, qtmr_channel_selection_t channel, qtmr_counting_mode_t clockSource ) [inline], [static]`

## Parameters

|                    |                                    |
|--------------------|------------------------------------|
| <i>base</i>        | Quad Timer peripheral base address |
| <i>channel</i>     | Quad Timer channel number          |
| <i>clockSource</i> | Quad Timer clock source            |

**37.4.16** `static void QTMR_StopTimer ( TMR_Type * base, qtmr_channel_selection_t channel ) [inline], [static]`

## Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |

**37.4.17** `void QTMR_EnableDma ( TMR_Type * base, qtmr_channel_selection_t channel, uint32_t mask )`

## Parameters

|                |                                                                                                         |
|----------------|---------------------------------------------------------------------------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address                                                                      |
| <i>channel</i> | Quad Timer channel number                                                                               |
| <i>mask</i>    | The DMA to enable. This is a logical OR of members of the enumeration <a href="#">qtmr_dma_enable_t</a> |

**37.4.18 void QTMR\_DisableDma ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, uint32\_t *mask* )**

Parameters

|                |                                                                                                         |
|----------------|---------------------------------------------------------------------------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address                                                                      |
| <i>channel</i> | Quad Timer channel number                                                                               |
| <i>mask</i>    | The DMA to enable. This is a logical OR of members of the enumeration <a href="#">qtmr_dma_enable_t</a> |

**37.4.19 void QTMR\_SetPwmOutputTogle ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, bool *idleStatus* )**

Note

When the PWM is set again, the counting needs to be restarted.

Parameters

|                   |                                                                                   |
|-------------------|-----------------------------------------------------------------------------------|
| <i>base</i>       | Quad Timer peripheral base address                                                |
| <i>channel</i>    | Quad Timer channel number                                                         |
| <i>idleStatus</i> | True: PWM output is high in idle status; false: PWM output is low in idle status. |

**37.4.20 static qtmr\_pwm\_out\_state\_t QTMR\_GetPwmOutputStatus ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel* ) [inline], [static]**

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |

Returns

Current channel output status.

**37.4.21** `uint8_t QTMR_GetPwmChannelStatus ( TMR_Type * base,  
qtmr_channel_selection_t channel )`

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |

Returns

Current channel dutycycle value.

**37.4.22** `void QTMR_SetPwmClockMode ( TMR_Type * base, qtmr_channel-  
_selection_t channel, qtmr_primary_count_source_t prescaler  
)`

Parameters

|                  |                                    |
|------------------|------------------------------------|
| <i>base</i>      | Quad Timer peripheral base address |
| <i>channel</i>   | Quad Timer channel number          |
| <i>prescaler</i> | Set prescaler value                |

# Chapter 38

## RTWDOG: 32-bit Watchdog Timer

### 38.1 Overview

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

### 38.2 Typical use case

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

### Data Structures

- struct `rtwdog_work_mode_t`  
*Defines RTWDOG work mode. [More...](#)*
- struct `rtwdog_config_t`  
*Describes RTWDOG configuration structure. [More...](#)*

### Enumerations

- enum `rtwdog_clock_source_t` {  
    `kRTWDOG_ClockSource0` = 0U,  
    `kRTWDOG_ClockSource1` = 1U,  
    `kRTWDOG_ClockSource2` = 2U,  
    `kRTWDOG_ClockSource3` = 3U }  
*Describes RTWDOG clock source.*
- enum `rtwdog_clock_prescaler_t` {  
    `kRTWDOG_ClockPrescalerDivide1` = 0x0U,  
    `kRTWDOG_ClockPrescalerDivide256` = 0x1U }  
*Describes the selection of the clock prescaler.*
- enum `rtwdog_test_mode_t` {  
    `kRTWDOG_TestModeDisabled` = 0U,  
    `kRTWDOG_UserModeEnabled` = 1U,  
    `kRTWDOG_LowByteTest` = 2U,  
    `kRTWDOG_HighByteTest` = 3U }  
*Describes RTWDOG test mode.*
- enum `_rtwdog_interrupt_enable_t` { `kRTWDOG_InterruptEnable` = RTWDOG\_CS\_INT\_MASK }  
*RTWDOG interrupt configuration structure.*
- enum `_rtwdog_status_flags_t` {  
    `kRTWDOG_RunningFlag` = RTWDOG\_CS\_EN\_MASK,  
    `kRTWDOG_InterruptFlag` = RTWDOG\_CS\_FLG\_MASK }  
*RTWDOG status flags.*

## Unlock sequence

- #define `WDOG_FIRST_WORD_OF_UNLOCK` (RTWDOG\_UPDATE\_KEY & 0xFFFFU)  
*First word of unlock sequence.*
- #define `WDOG_SECOND_WORD_OF_UNLOCK` ((RTWDOG\_UPDATE\_KEY >> 16U) & 0xFFFFU)  
*Second word of unlock sequence.*

## Refresh sequence

- #define `WDOG_FIRST_WORD_OF_REFRESH` (RTWDOG\_REFRESH\_KEY & 0xFFFFU)  
*First word of refresh sequence.*
- #define `WDOG_SECOND_WORD_OF_REFRESH` ((RTWDOG\_REFRESH\_KEY >> 16U) & 0xFFFFU)  
*Second word of refresh sequence.*

## Driver version

- #define `FSL_RTWDOG_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 2))  
*RTWDOG driver version 2.1.2.*

## RTWDOG Initialization and De-initialization

- void `RTWDOG_GetDefaultConfig` (rtwdog\_config\_t \*config)  
*Initializes the RTWDOG configuration structure.*
- void `RTWDOG_Init` (RTWDOG\_Type \*base, const rtwdog\_config\_t \*config)  
*Initializes the RTWDOG module.*
- void `RTWDOG_Deinit` (RTWDOG\_Type \*base)  
*De-initializes the RTWDOG module.*

## RTWDOG functional Operation

- static void `RTWDOG_Enable` (RTWDOG\_Type \*base)  
*Enables the RTWDOG module.*
- static void `RTWDOG_Disable` (RTWDOG\_Type \*base)  
*Disables the RTWDOG module.*
- static void `RTWDOG_EnableInterrupts` (RTWDOG\_Type \*base, uint32\_t mask)  
*Enables the RTWDOG interrupt.*
- static void `RTWDOG_DisableInterrupts` (RTWDOG\_Type \*base, uint32\_t mask)  
*Disables the RTWDOG interrupt.*
- static uint32\_t `RTWDOG_GetStatusFlags` (RTWDOG\_Type \*base)  
*Gets the RTWDOG all status flags.*
- static void `RTWDOG_EnableWindowMode` (RTWDOG\_Type \*base, bool enable)  
*Enables/disables the window mode.*
- static uint32\_t `RTWDOG_CountToMesec` (RTWDOG\_Type \*base, uint32\_t count, uint32\_t clock-FreqInHz)  
*Converts raw count value to millisecond.*
- void `RTWDOG_ClearStatusFlags` (RTWDOG\_Type \*base, uint32\_t mask)  
*Clears the RTWDOG flag.*
- static void `RTWDOG_SetTimeoutValue` (RTWDOG\_Type \*base, uint16\_t timeoutCount)

- *Sets the RTWDOG timeout value.*
- static void [RTWDOG\\_SetWindowValue](#) (RTWDOG\_Type \*base, uint16\_t windowValue)
  - *Sets the RTWDOG window value.*
- `__STATIC_FORCEINLINE` void [RTWDOG\\_Unlock](#) (RTWDOG\_Type \*base)
  - *Unlocks the RTWDOG register written.*
- static void [RTWDOG\\_Refresh](#) (RTWDOG\_Type \*base)
  - *Refreshes the RTWDOG timer.*
- static uint16\_t [RTWDOG\\_GetCounterValue](#) (RTWDOG\_Type \*base)
  - *Gets the RTWDOG counter value.*

## 38.3 Data Structure Documentation

### 38.3.1 struct rtwdog\_work\_mode\_t

#### Data Fields

- bool [enableWait](#)
  - *Enables or disables RTWDOG in wait mode.*
- bool [enableStop](#)
  - *Enables or disables RTWDOG in stop mode.*
- bool [enableDebug](#)
  - *Enables or disables RTWDOG in debug mode.*

### 38.3.2 struct rtwdog\_config\_t

#### Data Fields

- bool [enableRtwdog](#)
  - *Enables or disables RTWDOG.*
- [rtwdog\\_clock\\_source\\_t](#) [clockSource](#)
  - *Clock source select.*
- [rtwdog\\_clock\\_prescaler\\_t](#) [prescaler](#)
  - *Clock prescaler value.*
- [rtwdog\\_work\\_mode\\_t](#) [workMode](#)
  - *Configures RTWDOG work mode in debug stop and wait mode.*
- [rtwdog\\_test\\_mode\\_t](#) [testMode](#)
  - *Configures RTWDOG test mode.*
- bool [enableUpdate](#)
  - *Update write-once register enable.*
- bool [enableInterrupt](#)
  - *Enables or disables RTWDOG interrupt.*
- bool [enableWindowMode](#)
  - *Enables or disables RTWDOG window mode.*
- uint16\_t [windowValue](#)
  - *Window value.*
- uint16\_t [timeoutValue](#)
  - *Timeout value.*

## 38.4 Macro Definition Documentation

### 38.4.1 #define FSL\_RTWD0G\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 2))

## 38.5 Enumeration Type Documentation

### 38.5.1 enum rtwdog\_clock\_source\_t

Enumerator

*kRTWD0G\_ClockSource0* Clock source 0.  
*kRTWD0G\_ClockSource1* Clock source 1.  
*kRTWD0G\_ClockSource2* Clock source 2.  
*kRTWD0G\_ClockSource3* Clock source 3.

### 38.5.2 enum rtwdog\_clock\_prescaler\_t

Enumerator

*kRTWD0G\_ClockPrescalerDivide1* Divided by 1.  
*kRTWD0G\_ClockPrescalerDivide256* Divided by 256.

### 38.5.3 enum rtwdog\_test\_mode\_t

Enumerator

*kRTWD0G\_TestModeDisabled* Test Mode disabled.  
*kRTWD0G\_UserModeEnabled* User Mode enabled.  
*kRTWD0G\_LowByteTest* Test Mode enabled, only low byte is used.  
*kRTWD0G\_HighByteTest* Test Mode enabled, only high byte is used.

### 38.5.4 enum \_rtwdog\_interrupt\_enable\_t

This structure contains the settings for all of the RTWD0G interrupt configurations.

Enumerator

*kRTWD0G\_InterruptEnable* Interrupt is generated before forcing a reset.

### 38.5.5 enum\_rtwdog\_status\_flags\_t

This structure contains the RTWDOG status flags for use in the RTWDOG functions.

Enumerator

- kRTWDOG\_RunningFlag* Running flag, set when RTWDOG is enabled.
- kRTWDOG\_InterruptFlag* Interrupt flag, set when interrupt occurs.

## 38.6 Function Documentation

### 38.6.1 void RTWDOG\_GetDefaultConfig ( rtwdog\_config\_t \* config )

This function initializes the RTWDOG configuration structure to default values. The default values are:

```
* rtwdogConfig->enableRtwdog = true;
* rtwdogConfig->clockSource = kRTWDOG_ClockSource1;
* rtwdogConfig->prescaler = kRTWDOG_ClockPrescalerDivide1;
* rtwdogConfig->workMode.enableWait = true;
* rtwdogConfig->workMode.enableStop = false;
* rtwdogConfig->workMode.enableDebug = false;
* rtwdogConfig->testMode = kRTWDOG_TestModeDisabled;
* rtwdogConfig->enableUpdate = true;
* rtwdogConfig->enableInterrupt = false;
* rtwdogConfig->enableWindowMode = false;
* rtwdogConfig->windowValue = 0U;
* rtwdogConfig->timeoutValue = 0xFFFFU;
*
```

Parameters

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

See Also

[rtwdog\\_config\\_t](#)

### 38.6.2 void RTWDOG\_Init ( RTWDOG\_Type \* base, const rtwdog\_config\_t \* config )

This function initializes the RTWDOG. To reconfigure the RTWDOG without forcing a reset first, enableUpdate must be set to true in the configuration.

Example:

```
* rtwdog_config_t config;
* RTWDOG_GetDefaultConfig(&config);
* config.timeoutValue = 0x7ffU;
* config.enableUpdate = true;
* RTWDOG_Init(wdog_base, &config);
*
```



Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | RTWDOG peripheral base address.  |
| <i>config</i> | The configuration of the RTWDOG. |

### 38.6.3 void RTWDOG\_Deinit ( RTWDOG\_Type \* *base* )

This function shuts down the RTWDOG. Ensure that the WDOG\_CS.UPDATE is 1, which means that the register update is enabled.

Parameters

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

### 38.6.4 static void RTWDOG\_Enable ( RTWDOG\_Type \* *base* ) [inline], [static]

This function writes a value into the WDOG\_CS register to enable the RTWDOG. The WDOG\_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

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

### 38.6.5 static void RTWDOG\_Disable ( RTWDOG\_Type \* *base* ) [inline], [static]

This function writes a value into the WDOG\_CS register to disable the RTWDOG. The WDOG\_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

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

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

This function writes a value into the WDOG\_CS register to enable the RTWDOG interrupt. The WDOG\_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

|             |                                                                                                                                                                              |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | RTWDOG peripheral base address.                                                                                                                                              |
| <i>mask</i> | The interrupts to enable. The parameter can be a combination of the following source if defined: <ul style="list-style-type: none"> <li>• kRTWDOG_InterruptEnable</li> </ul> |

### 38.6.7 static void RTWDOG\_DisableInterrupts ( RTWDOG\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function writes a value into the WDOG\_CS register to disable the RTWDOG interrupt. The WDOG\_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

|             |                                                                                                                                                                                |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | RTWDOG peripheral base address.                                                                                                                                                |
| <i>mask</i> | The interrupts to disabled. The parameter can be a combination of the following source if defined: <ul style="list-style-type: none"> <li>• kRTWDOG_InterruptEnable</li> </ul> |

### 38.6.8 static uint32\_t RTWDOG\_GetStatusFlags ( RTWDOG\_Type \* *base* ) [inline], [static]

This function gets all status flags.

Example to get the running flag:

```
* uint32_t status;
* status = RTWDOG_GetStatusFlags(wdog_base) &
* kRTWDOG_RunningFlag;
*
```

## Parameters

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

## Returns

State of the status flag: asserted (true) or not-asserted (false).

## See Also

[\\_rtwdog\\_status\\_flags\\_t](#)

- true: related status flag has been set.
- false: related status flag is not set.

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

## Parameters

|               |                                               |
|---------------|-----------------------------------------------|
| <i>base</i>   | RTWDOG peripheral base address.               |
| <i>enable</i> | Enables(true) or disables(false) the feature. |

### 38.6.10 static uint32\_t RTWDOG\_CountToMesec ( RTWDOG\_Type \* *base*, uint32\_t *count*, uint32\_t *clockFreqInHz* ) [inline], [static]

Note that if the clock frequency is too high the timeout period can be less than 1 ms. In this case this api will return 0 value.

## Parameters

|                      |                                                |
|----------------------|------------------------------------------------|
| <i>base</i>          | RTWDOG peripheral base address.                |
| <i>count</i>         | Raw count value.                               |
| <i>clockFreqInHz</i> | The frequency of the clock source RTWDOG uses. |

### 38.6.11 void RTWDOG\_ClearStatusFlags ( RTWDOG\_Type \* *base*, uint32\_t *mask* )

This function clears the RTWDOG status flag.

Example to clear an interrupt flag:

```
* RTWDOG_ClearStatusFlags (wdog_base,
 kRTWDOG_InterruptFlag);
*
```

## Parameters

|             |                                                                                                                                                                    |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | RTWDOG peripheral base address.                                                                                                                                    |
| <i>mask</i> | The status flags to clear. The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• kRTWDOG_InterruptFlag</li> </ul> |

### 38.6.12 static void RTWDOG\_SetTimeoutValue ( RTWDOG\_Type \* *base*, uint16\_t *timeoutCount* ) [inline], [static]

This function writes a timeout value into the WDOG\_TOVAL register. The WDOG\_TOVAL register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

## Parameters

|                     |                                                    |
|---------------------|----------------------------------------------------|
| <i>base</i>         | RTWDOG peripheral base address                     |
| <i>timeoutCount</i> | RTWDOG timeout value, count of RTWDOG clock ticks. |

### 38.6.13 static void RTWDOG\_SetWindowValue ( RTWDOG\_Type \* *base*, uint16\_t *windowValue* ) [inline], [static]

This function writes a window value into the WDOG\_WIN register. The WDOG\_WIN register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

## Parameters

|                    |                                 |
|--------------------|---------------------------------|
| <i>base</i>        | RTWDOG peripheral base address. |
| <i>windowValue</i> | RTWDOG window value.            |

### 38.6.14 \_\_STATIC\_FORCEINLINE void RTWDOG\_Unlock ( RTWDOG\_Type \* *base* )

This function unlocks the RTWDOG register written.

Before starting the unlock sequence and following the configuration, disable the global interrupts. Otherwise, an interrupt could effectively invalidate the unlock sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

Parameters

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

### 38.6.15 **static void RTWDOG\_Refresh ( RTWDOG\_Type \* *base* ) [inline], [static]**

This function feeds the RTWDOG. This function should be called before the Watchdog timer is in timeout. Otherwise, a reset is asserted.

Parameters

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

### 38.6.16 **static uint16\_t RTWDOG\_GetCounterValue ( RTWDOG\_Type \* *base* ) [inline], [static]**

This function gets the RTWDOG counter value.

Parameters

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

Returns

Current RTWDOG counter value.

# Chapter 39

## SAI: Serial Audio Interface

### 39.1 Overview

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

SAI driver includes functional APIs and transactional APIs.

Functional APIs target low-level APIs. Functional APIs can be used for SAI initialization, configuration and operation, and for optimization and customization purposes. Using the functional API requires the knowledge of the SAI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SAI functional operation groups provide the functional API set.

Transactional APIs target high-level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the `sai_handle_t` as the first parameter. Initialize the handle by calling the [SAI\\_TransferTxCreateHandle\(\)](#) or [SAI\\_TransferRxCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [SAI\\_TransferSendNonBlocking\(\)](#) and [SAI\\_TransferReceiveNonBlocking\(\)](#) set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_SAI_TxIdle` and `kStatus_SAI_RxIdle` status.

### 39.2 Typical configurations

#### Bit width configuration

SAI driver support 8/16/24/32bits stereo/mono raw audio data transfer. SAI EDMA driver support 8/16/32bits stereo/mono raw audio data transfer, since the EDMA doesn't support 24bit data width, so application should pre-convert the 24bit data to 32bit. SAI DMA driver support 8/16/32bits stereo/mono raw audio data transfer, since the EDMA doesn't support 24bit data width, so application should pre-convert the 24bit data to 32bit. SAI SDMA driver support 8/16/24/32bits stereo/mono raw audio data transfer.

#### Frame configuration

SAI driver support I2S, DSP, Left justified, Right justified, TDM mode. Application can call the api directly: `SAI_GetClassicI2SConfig` `SAI_GetLeftJustifiedConfig` `SAI_GetRightJustifiedConfig` `SAI_GetTDMConfig` `SAI_GetDSPConfig`

## 39.3 Typical use case

### 39.3.1 SAI Send/receive using an interrupt method

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

### 39.3.2 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](#)

## 39.4 SAI Driver

### 39.4.1 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_bclk_source_t {
    kSAI_BclkSourceBusclk = 0x0U,
    kSAI_BclkSourceMclkOption1 = 0x1U,
    kSAI_BclkSourceMclkOption2 = 0x2U,
    kSAI_BclkSourceMclkOption3 = 0x3U,
    kSAI_BclkSourceMclkDiv = 0x1U,
    kSAI_BclkSourceOtherSai0 = 0x2U,
    kSAI_BclkSourceOtherSai1 = 0x3U }
    Bit clock source.
• enum {
    kSAI_WordStartInterruptEnable,
    kSAI_SyncErrorInterruptEnable = I2S_TCSR_SEIE_MASK,
    kSAI_FIFOWarningInterruptEnable = I2S_TCSR_FWIE_MASK,
    kSAI_FIFOErrorInterruptEnable = I2S_TCSR_FEIE_MASK,
    kSAI_FIFORequestInterruptEnable = I2S_TCSR_FRIE_MASK }
    _sai_interrupt_enable_t, The SAI interrupt enable flag
• enum {
    kSAI_FIFOWarningDMAEnable = I2S_TCSR_FWDE_MASK,
    kSAI_FIFORequestDMAEnable = I2S_TCSR_FRDE_MASK }
    _sai_dma_enable_t, The DMA request sources
• enum {
    kSAI_WordStartFlag = I2S_TCSR_WSF_MASK,
    kSAI_SyncErrorFlag = I2S_TCSR_SEF_MASK,
    kSAI_FIFOErrorFlag = I2S_TCSR_FEF_MASK,
    kSAI_FIFORequestFlag = I2S_TCSR_FRF_MASK,
    kSAI_FIFOWarningFlag = I2S_TCSR_FWF_MASK }
    _sai_flags, The SAI status flag
• enum sai_reset_type_t {
    kSAI_ResetTypeSoftware = I2S_TCSR_SR_MASK,
    kSAI_ResetTypeFIFO = I2S_TCSR_FR_MASK,
    kSAI_ResetAll = I2S_TCSR_SR_MASK | I2S_TCSR_FR_MASK }
    The reset type.
• enum sai_fifo_packing_t {
    kSAI_FifoPackingDisabled = 0x0U,
    kSAI_FifoPacking8bit = 0x2U,
    kSAI_FifoPacking16bit = 0x3U }
    The SAI packing mode The mode includes 8 bit and 16 bit packing.
• enum sai_sample_rate_t {

```

```

kSAI_SampleRate8KHz = 8000U,
kSAI_SampleRate11025Hz = 11025U,
kSAI_SampleRate12KHz = 12000U,
kSAI_SampleRate16KHz = 16000U,
kSAI_SampleRate22050Hz = 22050U,
kSAI_SampleRate24KHz = 24000U,
kSAI_SampleRate32KHz = 32000U,
kSAI_SampleRate44100Hz = 44100U,
kSAI_SampleRate48KHz = 48000U,
kSAI_SampleRate96KHz = 96000U,
kSAI_SampleRate192KHz = 192000U,
kSAI_SampleRate384KHz = 384000U }

```

Audio sample rate.

- enum `sai_word_width_t` {


```

kSAI_WordWidth8bits = 8U,
kSAI_WordWidth16bits = 16U,
kSAI_WordWidth24bits = 24U,
kSAI_WordWidth32bits = 32U }

```

Audio word width.
- enum `sai_data_pin_state_t` {


```

kSAI_DataPinStateTriState,
kSAI_DataPinStateOutputZero = 1U }

```

sai data pin state definition
- enum `sai_fifo_combine_t` {


```

kSAI_FifoCombineDisabled = 0U,
kSAI_FifoCombineModeEnabledOnRead,
kSAI_FifoCombineModeEnabledOnWrite,
kSAI_FifoCombineModeEnabledOnReadWrite }

```

sai fifo combine mode definition
- 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, 8)`)
Version 2.3.8.

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 receiver configurations.*
- void [SAI_GetClassicI2SConfig](#) ([sai_transceiver_t](#) *config, [sai_word_width_t](#) bitWidth, [sai_mono_stereo_t](#) mode, [uint32_t](#) saiChannelMask)
- *Get classic I2S mode configurations.*
- void [SAI_GetLeftJustifiedConfig](#) ([sai_transceiver_t](#) *config, [sai_word_width_t](#) bitWidth, [sai_mono_stereo_t](#) mode, [uint32_t](#) saiChannelMask)
- *Get left justified mode configurations.*
- void [SAI_GetRightJustifiedConfig](#) ([sai_transceiver_t](#) *config, [sai_word_width_t](#) bitWidth, [sai_mono_stereo_t](#) mode, [uint32_t](#) saiChannelMask)
- *Get right justified mode configurations.*
- void [SAI_GetTDMConfig](#) ([sai_transceiver_t](#) *config, [sai_frame_sync_len_t](#) frameSyncWidth, [sai_word_width_t](#) bitWidth, [uint32_t](#) dataWordNum, [uint32_t](#) saiChannelMask)
- *Get TDM mode configurations.*
- void [SAI_GetDSPConfig](#) ([sai_transceiver_t](#) *config, [sai_frame_sync_len_t](#) frameSyncWidth, [sai_word_width_t](#) bitWidth, [sai_mono_stereo_t](#) mode, [uint32_t](#) saiChannelMask)
- *Get DSP mode configurations.*

Status

- static [uint32_t SAI_TxGetStatusFlag](#) (I2S_Type *base)
- *Gets the SAI Tx status flag state.*
- static void [SAI_TxClearStatusFlags](#) (I2S_Type *base, [uint32_t](#) mask)
- *Clears the SAI Tx status flag state.*
- static [uint32_t SAI_RxGetStatusFlag](#) (I2S_Type *base)
- *Gets the SAI Rx status flag state.*
- static void [SAI_RxClearStatusFlags](#) (I2S_Type *base, [uint32_t](#) mask)
- *Clears the SAI Rx status flag state.*
- void [SAI_TxSoftwareReset](#) (I2S_Type *base, [sai_reset_type_t](#) resetType)
- *Do software reset or FIFO reset .*
- void [SAI_RxSoftwareReset](#) (I2S_Type *base, [sai_reset_type_t](#) resetType)
- *Do software reset or FIFO reset .*
- void [SAI_TxSetChannelFIFOMask](#) (I2S_Type *base, [uint8_t](#) mask)
- *Set the Tx channel FIFO enable mask.*
- void [SAI_RxSetChannelFIFOMask](#) (I2S_Type *base, [uint8_t](#) mask)
- *Set the Rx channel FIFO enable mask.*
- void [SAI_TxSetDataOrder](#) (I2S_Type *base, [sai_data_order_t](#) order)
- *Set the Tx data order.*
- void [SAI_RxSetDataOrder](#) (I2S_Type *base, [sai_data_order_t](#) order)

- *Set the Rx data order.*
- void [SAI_TxSetBitClockPolarity](#) (I2S_Type *base, [sai_clock_polarity_t](#) polarity)
- *Set the Tx data order.*
- void [SAI_RxSetBitClockPolarity](#) (I2S_Type *base, [sai_clock_polarity_t](#) polarity)
- *Set the Rx data order.*
- void [SAI_TxSetFrameSyncPolarity](#) (I2S_Type *base, [sai_clock_polarity_t](#) polarity)
- *Set the Tx data order.*
- void [SAI_RxSetFrameSyncPolarity](#) (I2S_Type *base, [sai_clock_polarity_t](#) polarity)
- *Set the Rx data order.*
- void [SAI_TxSetFIFOPacking](#) (I2S_Type *base, [sai_fifo_packing_t](#) pack)
- *Set Tx FIFO packing feature.*
- void [SAI_RxSetFIFOPacking](#) (I2S_Type *base, [sai_fifo_packing_t](#) pack)
- *Set Rx FIFO packing feature.*
- static void [SAI_TxSetFIFOErrorContinue](#) (I2S_Type *base, bool isEnabled)
- *Set Tx FIFO error continue.*
- static void [SAI_RxSetFIFOErrorContinue](#) (I2S_Type *base, bool isEnabled)
- *Set Rx FIFO error continue.*

Interrupts

- static void [SAI_TxEnableInterrupts](#) (I2S_Type *base, uint32_t mask)
- *Enables the SAI Tx interrupt requests.*
- static void [SAI_RxEnableInterrupts](#) (I2S_Type *base, uint32_t mask)
- *Enables the SAI Rx interrupt requests.*
- static void [SAI_TxDisableInterrupts](#) (I2S_Type *base, uint32_t mask)
- *Disables the SAI Tx interrupt requests.*
- static void [SAI_RxDisableInterrupts](#) (I2S_Type *base, uint32_t mask)
- *Disables the SAI Rx interrupt requests.*

DMA Control

- static void [SAI_TxEnableDMA](#) (I2S_Type *base, uint32_t mask, bool enable)
- *Enables/disables the SAI Tx DMA requests.*
- static void [SAI_RxEnableDMA](#) (I2S_Type *base, uint32_t mask, bool enable)
- *Enables/disables the SAI Rx DMA requests.*
- static uintptr_t [SAI_TxGetDataRegisterAddress](#) (I2S_Type *base, uint32_t channel)
- *Gets the SAI Tx data register address.*
- static uintptr_t [SAI_RxGetDataRegisterAddress](#) (I2S_Type *base, uint32_t channel)
- *Gets the SAI Rx data register address.*

Bus Operations

- void [SAI_TxSetFormat](#) (I2S_Type *base, [sai_transfer_format_t](#) *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
- *Configures the SAI Tx audio format.*
- void [SAI_RxSetFormat](#) (I2S_Type *base, [sai_transfer_format_t](#) *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)

- Configures the SAI Rx audio format.*

void [SAI_WriteBlocking](#) (I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data using a blocking method.
- void [SAI_WriteMultiChannelBlocking](#) (I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data to multi channel using a blocking method.
- static void [SAI_WriteData](#) (I2S_Type *base, uint32_t channel, uint32_t data)

Writes data into SAI FIFO.
- void [SAI_ReadBlocking](#) (I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives data using a blocking method.
- void [SAI_ReadMultiChannelBlocking](#) (I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives multi channel data using a blocking method.
- static uint32_t [SAI_ReadData](#) (I2S_Type *base, uint32_t channel)

Reads data from the SAI FIFO.

Transactional

- void [SAI_TransferTxCreateHandle](#) (I2S_Type *base, sai_handle_t *handle, sai_transfer_callback_t callback, void *userData)

Initializes the SAI Tx handle.
- void [SAI_TransferRxCreateHandle](#) (I2S_Type *base, sai_handle_t *handle, sai_transfer_callback_t callback, void *userData)

Initializes the SAI Rx handle.
- void [SAI_TransferTxSetConfig](#) (I2S_Type *base, sai_handle_t *handle, sai_transceiver_t *config)

SAI transmitter transfer configurations.
- void [SAI_TransferRxSetConfig](#) (I2S_Type *base, sai_handle_t *handle, sai_transceiver_t *config)

SAI receiver transfer configurations.
- status_t [SAI_TransferTxSetFormat](#) (I2S_Type *base, sai_handle_t *handle, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)

Configures the SAI Tx audio format.
- status_t [SAI_TransferRxSetFormat](#) (I2S_Type *base, sai_handle_t *handle, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)

Configures the SAI Rx audio format.
- status_t [SAI_TransferSendNonBlocking](#) (I2S_Type *base, sai_handle_t *handle, sai_transfer_t *xfer)

Performs an interrupt non-blocking send transfer on SAI.
- status_t [SAI_TransferReceiveNonBlocking](#) (I2S_Type *base, sai_handle_t *handle, sai_transfer_t *xfer)

Performs an interrupt non-blocking receive transfer on SAI.
- status_t [SAI_TransferGetSendCount](#) (I2S_Type *base, sai_handle_t *handle, size_t *count)

Gets a set byte count.
- status_t [SAI_TransferGetReceiveCount](#) (I2S_Type *base, sai_handle_t *handle, size_t *count)

Gets a received byte count.
- void [SAI_TransferAbortSend](#) (I2S_Type *base, sai_handle_t *handle)

Aborts the current send.
- void [SAI_TransferAbortReceive](#) (I2S_Type *base, sai_handle_t *handle)

- *Aborts the current IRQ receive.*
- void [SAI_TransferTerminateSend](#) (I2S_Type *base, sai_handle_t *handle)
Terminate all SAI send.
- void [SAI_TransferTerminateReceive](#) (I2S_Type *base, sai_handle_t *handle)
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)
Tx interrupt handler.

39.4.2 Data Structure Documentation

39.4.2.1 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_bclk_source_t bclkSource](#)
Bit Clock source.
- [sai_master_slave_t masterSlave](#)
Master or slave.

39.4.2.2 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.
- 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.

Field Documentation

(1) bool `sai_transfer_format_t::isFrameSyncCompact`

39.4.2.3 struct `sai_master_clock_t`

Data Fields

- bool `mclkOutputEnable`
master clock output enable
- uint32_t `mclkHz`
target mclk frequency
- uint32_t `mclkSourceClkHz`
mclk source frequency

39.4.2.4 struct `sai_fifo_t`

Data Fields

- bool `fifoContinueOnError`
fifo continues when error occur
- `sai_fifo_combine_t` `fifoCombine`
fifo combine mode
- `sai_fifo_packing_t` `fifoPacking`
fifo packing mode
- uint8_t `fifoWatermark`
fifo watermark

39.4.2.5 struct `sai_bit_clock_t`

Data Fields

- bool `bclkSrcSwap`
bit clock source swap
- bool `bclkInputDelay`
bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time .
- `sai_clock_polarity_t` `bclkPolarity`
bit clock polarity
- `sai_bclk_source_t` `bclkSource`
bit Clock source

Field Documentation

(1) bool sai_bit_clock_t::bclkInputDelay

39.4.2.6 struct sai_frame_sync_t

Data Fields

- uint8_t [frameSyncWidth](#)
frame sync width in number of bit clocks
- bool [frameSyncEarly](#)
TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame.
- bool [frameSyncGenerateOnDemand](#)
internal frame sync is generated when FIFO warning flag is clear
- [sai_clock_polarity_t](#) [frameSyncPolarity](#)
frame sync polarity

39.4.2.7 struct sai_serial_data_t

Data Fields

- [sai_data_pin_state_t](#) [dataMode](#)
sai data pin state when slots masked or channel disabled
- [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

39.4.2.8 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.

39.4.2.9 struct `sai_transfer_t`

Data Fields

- `uint8_t *` `data`
Data start address to transfer.
- `size_t` `dataSize`
Transfer size.

Field Documentation

(1) `uint8_t*` `sai_transfer_t::data`

(2) `size_t` `sai_transfer_t::dataSize`

39.4.2.10 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.

39.4.3 Macro Definition Documentation

39.4.3.1 #define SAI_XFER_QUEUE_SIZE (4U)

39.4.4 Enumeration Type Documentation

39.4.4.1 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.

39.4.4.2 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

39.4.4.3 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.

39.4.4.4 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

39.4.4.5 enum sai_mono_stereo_t

Enumerator

kSAI_Stereo Stereo sound.
kSAI_MonoRight Only Right channel have sound.
kSAI_MonoLeft Only left channel have sound.

39.4.4.6 enum sai_data_order_t

Enumerator

kSAI_DataLSB LSB bit transferred first.
kSAI_DataMSB MSB bit transferred first.

39.4.4.7 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.

39.4.4.8 enum sai_sync_mode_t

Enumerator

- kSAI_ModeAsync* Asynchronous mode.
- kSAI_ModeSync* Synchronous mode (with receiver or transmit)

39.4.4.9 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.

39.4.4.10 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.

39.4.4.11 anonymous enum

Enumerator

- kSAI_FIFOWarningDMAEnable* FIFO warning caused by the DMA request.
- kSAI_FIFORequestDMAEnable* FIFO request caused by the DMA request.

39.4.4.12 anonymous enum

Enumerator

- kSAI_WordStartFlag* Word start flag, means the first word in a frame detected.
- kSAI_SyncErrorFlag* Sync error flag, means the sync error is detected.
- kSAI_FIFOErrorFlag* FIFO error flag.
- kSAI_FIFORequestFlag* FIFO request flag.
- kSAI_FIFOWarningFlag* FIFO warning flag.

39.4.4.13 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.

39.4.4.14 enum sai_fifo_packing_t

Enumerator

kSAI_FifoPackingDisabled Packing disabled.
kSAI_FifoPacking8bit 8 bit packing enabled
kSAI_FifoPacking16bit 16bit packing enabled

39.4.4.15 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.

39.4.4.16 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.

39.4.4.17 enum sai_data_pin_state_t

Enumerator

kSAI_DataPinStateTriState transmit data pins are tri-stated when slots are masked or channels are disabled

kSAI_DataPinStateOutputZero transmit data pins are never tri-stated and will output zero when slots are masked or channel disabled

39.4.4.18 enum sai_fifo_combine_t

Enumerator

kSAI_FifoCombineDisabled sai fifo combine mode disabled

kSAI_FifoCombineModeEnabledOnRead sai fifo combine mode enabled on FIFO reads

kSAI_FifoCombineModeEnabledOnWrite sai fifo combine mode enabled on FIFO write

kSAI_FifoCombineModeEnabledOnReadWrite sai fifo combined mode enabled on FIFO read/writes

39.4.4.19 enum sai_transceiver_type_t

Enumerator

kSAI_Transmitter sai transmitter

kSAI_Receiver sai receiver

39.4.4.20 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.

39.4.5 Function Documentation**39.4.5.1 void SAI_TxInit (I2S_Type * *base*, const sai_config_t * *config*)**

Deprecated Do not use this function. It has been superseded 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.

39.4.5.2 void SAI_RxInit (I2S_Type * *base*, const sai_config_t * *config*)

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

Ungates the SAI clock, resets the module, and configures the SAI Rx with a configuration structure. The configuration structure can be custom filled or set with default values by [SAI_RxGetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAI module can cause a hard fault because the clock is not enabled.

Parameters

<i>base</i>	SAI base pointer
<i>config</i>	SAI configuration structure.

39.4.5.3 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
---------------	---

39.4.5.4 void SAI_RxGetDefaultConfig (sai_config_t * config)

Deprecated Do not use this function. It has been superseded by [SAI_GetClassicI2SConfig](#), [SAI_GetLeft-JustifiedConfig](#) , [SAI_GetRightJustifiedConfig](#), [SAI_GetDSPConfig](#), [SAI_GetTDMConfig](#)

This API initializes the configuration structure for use in SAI_RxConfig(). The initialized structure can remain unchanged in SAI_RxConfig() or it can be modified before calling SAI_RxConfig(). This is an example.

```
sai_config_t config;
SAI_RxGetDefaultConfig(&config);
```

Parameters

<i>config</i>	pointer to master configuration structure
---------------	---

39.4.5.5 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.
-------------	-------------------

39.4.5.6 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.
-------------	-------------------

39.4.5.7 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
-------------	------------------

39.4.5.8 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
-------------	------------------

39.4.5.9 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.

39.4.5.10 void SAI_RxEnable (I2S_Type * *base*, bool *enable*)

Parameters

<i>base</i>	SAI base pointer.
<i>enable</i>	True means enable SAI Rx, false means disable.

39.4.5.11 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.
--------------------	-------------------------------

39.4.5.12 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.

39.4.5.13 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.

39.4.5.14 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.

39.4.5.15 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.

39.4.5.16 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.

39.4.5.17 void SAI_TxSetBitclockConfig (I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_bit_clock_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>masterSlave</i>	master or slave.
<i>config</i>	bit clock other configurations, can be NULL in slave mode.

39.4.5.18 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.

39.4.5.19 void SAI_SetMasterClockConfig (I2S_Type * *base*, sai_master_clock_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	master clock configurations.

39.4.5.20 void SAI_TxSetFifoConfig (I2S_Type * *base*, sai_fifo_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	fifo configurations.

39.4.5.21 void SAI_RxSetFifoConfig (I2S_Type * *base*, sai_fifo_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	fifo configurations.

39.4.5.22 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.

39.4.5.23 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.

39.4.5.24 void SAI_TxSetSerialDataConfig (I2S_Type * *base*, sai_serial_data_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	serial data configurations.

39.4.5.25 void SAI_RxSetSerialDataConfig (I2S_Type * *base*, sai_serial_data_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	serial data configurations.

39.4.5.26 void SAI_TxSetConfig (I2S_Type * *base*, sai_transceiver_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	transmitter configurations.

39.4.5.27 void SAI_RxSetConfig (I2S_Type * *base*, sai_transceiver_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	receiver configurations.

39.4.5.28 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>saiChannel-Mask</i>	mask value of the channel to be enable.

39.4.5.29 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.

39.4.5.30 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.

39.4.5.31 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.

39.4.5.32 void SAI_GetDSPConfig (sai_transceiver_t * *config*, sai_frame_sync_len_t *frameSyncWidth*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask*)

Note

DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
* SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth,
    kSAI_Stereo, channelMask)
* config->frameSync.frameSyncEarly = true;
* SAI_TxSetConfig(base, config)
*
```

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
* SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth,
    kSAI_Stereo, channelMask)
* SAI_TxSetConfig(base, config)
*
```

Parameters

<i>config</i>	transceiver configurations.
<i>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.

```
39.4.5.33 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.

39.4.5.34 `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"> • kSAI_WordStartFlag • kSAI_SyncErrorFlag • kSAI_FIFOErrorFlag

39.4.5.35 `static uint32_t SAI_RxGetStatusFlag (I2S_Type * base)` **[inline],**
[static]

Parameters

<i>base</i>	SAI base pointer
-------------	------------------

Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

39.4.5.36 `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"> • kSAI_WordStartFlag • kSAI_SyncErrorFlag • kSAI_FIFOErrorFlag

39.4.5.37 void SAI_TxSoftwareReset (I2S_Type * *base*, sai_reset_type_t *resetType*)

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

<i>base</i>	SAI base pointer
<i>resetType</i>	Reset type, FIFO reset or software reset

39.4.5.38 void SAI_RxSoftwareReset (I2S_Type * *base*, sai_reset_type_t *resetType*)

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

<i>base</i>	SAI base pointer
<i>resetType</i>	Reset type, FIFO reset or software reset

39.4.5.39 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.

39.4.5.40 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.

39.4.5.41 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

39.4.5.42 void SAI_RxSetDataOrder (I2S_Type * *base*, sai_data_order_t *order*)

Parameters

<i>base</i>	SAI base pointer
<i>order</i>	Data order MSB or LSB

39.4.5.43 void SAI_TxSetBitClockPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

<i>base</i>	SAI base pointer
<i>polarity</i>	

39.4.5.44 void SAI_RxSetBitClockPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

<i>base</i>	SAI base pointer
<i>polarity</i>	

39.4.5.45 void SAI_TxSetFrameSyncPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

<i>base</i>	SAI base pointer
<i>polarity</i>	

39.4.5.46 void SAI_RxSetFrameSyncPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

<i>base</i>	SAI base pointer
<i>polarity</i>	

39.4.5.47 void SAI_TxSetFIFOPacking (I2S_Type * *base*, sai_fifo_packing_t *pack*)

Parameters

<i>base</i>	SAI base pointer.
<i>pack</i>	FIFO pack type. It is element of sai_fifo_packing_t.

39.4.5.48 void SAI_RxSetFIFOPacking (I2S_Type * *base*, sai_fifo_packing_t *pack*)

Parameters

<i>base</i>	SAI base pointer.
<i>pack</i>	FIFO pack type. It is element of sai_fifo_packing_t.

**39.4.5.49 static void SAI_TxSetFIFOErrorContinue (I2S_Type * *base*, bool *isEnabled*)
[inline], [static]**

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

Parameters

<i>base</i>	SAI base pointer.
<i>isEnabled</i>	Is FIFO error continue enabled, true means enable, false means disable.

**39.4.5.50 static void SAI_RxSetFIFOErrorContinue (I2S_Type * *base*, bool *isEnabled*)
[inline], [static]**

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

<i>base</i>	SAI base pointer.
<i>isEnabled</i>	Is FIFO error continue enabled, true means enable, false means disable.

**39.4.5.51 static void SAI_TxEnableInterrupts (I2S_Type * *base*, uint32_t *mask*)
[inline], [static]**

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

**39.4.5.52 static void SAI_RxEnableInterrupts (I2S_Type * *base*, uint32_t *mask*)
[inline], [static]**

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

39.4.5.53 static void SAI_TxDisableInterrupts (I2S_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

39.4.5.54 static void SAI_RxDisableInterrupts (I2S_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

39.4.5.55 `static void SAI_TxEnableDMA (I2S_Type * base, uint32_t mask, bool enable)`
`[inline], [static]`

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	DMA source The parameter can be combination of the following sources if defined. <ul style="list-style-type: none"> • kSAI_FIFOWarningDMAEnable • kSAI_FIFORequestDMAEnable
<i>enable</i>	True means enable DMA, false means disable DMA.

39.4.5.56 `static void SAI_RxEnableDMA (I2S_Type * base, uint32_t mask, bool enable) [inline], [static]`

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	DMA source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> • kSAI_FIFOWarningDMAEnable • kSAI_FIFORequestDMAEnable
<i>enable</i>	True means enable DMA, false means disable DMA.

39.4.5.57 `static uintptr_t SAI_TxGetDataRegisterAddress (I2S_Type * base, uint32_t channel) [inline], [static]`

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Which data channel used.

Returns

data register address.

39.4.5.58 `static uintptr_t SAI_RxGetDataRegisterAddress (I2S_Type * base, uint32_t channel) [inline], [static]`

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Which data channel used.

Returns

data register address.

39.4.5.59 void SAI_TxSetFormat (I2S_Type * *base*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

Deprecated Do not use this function. It has been superceded by [SAI_TxSetConfig](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

<i>base</i>	SAI base pointer.
<i>format</i>	Pointer to the SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If the bit clock source is a master clock, this value should equal the masterClockHz.

39.4.5.60 void SAI_RxSetFormat (I2S_Type * *base*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

Deprecated Do not use this function. It has been superceded by [SAI_RxSetConfig](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

<i>base</i>	SAI base pointer.
<i>format</i>	Pointer to the SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If the bit clock source is a master clock, this value should equal the masterClockHz.

39.4.5.61 void SAI_WriteBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be written.
<i>size</i>	Bytes to be written.

39.4.5.62 void SAI_WriteMultiChannelBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *channelMask*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>channelMask</i>	channel mask.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be written.
<i>size</i>	Bytes to be written.

**39.4.5.63 static void SAI_WriteData (I2S_Type * *base*, uint32_t *channel*, uint32_t *data*)
[inline], [static]**

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>data</i>	Data needs to be written.

39.4.5.64 void SAI_ReadBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be read.
<i>size</i>	Bytes to be read.

39.4.5.65 void SAI_ReadMultiChannelBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *channelMask*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>channelMask</i>	channel mask.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be read.
<i>size</i>	Bytes to be read.

**39.4.5.66 static uint32_t SAI_ReadData (I2S_Type * *base*, uint32_t *channel*)
[inline], [static]**

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.

Returns

Data in SAI FIFO.

39.4.5.67 void SAI_TransferTxCreateHandle (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_callback_t *callback*, void * *userData*)

This function initializes the Tx handle for the SAI Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI handle pointer.
<i>callback</i>	Pointer to the user callback function.
<i>userData</i>	User parameter passed to the callback function

39.4.5.68 void SAI_TransferRxCreateHandle (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_callback_t *callback*, void * *userData*)

This function initializes the Rx handle for the SAI Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>callback</i>	Pointer to the user callback function.
<i>userData</i>	User parameter passed to the callback function.

39.4.5.69 void SAI_TransferTxSetConfig (I2S_Type * *base*, sai_handle_t * *handle*, sai_transceiver_t * *config*)

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>config</i>	transmitter configurations.

39.4.5.70 void SAI_TransferRxSetConfig (I2S_Type * *base*, sai_handle_t * *handle*, sai_transceiver_t * *config*)

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>config</i>	receiver configurations.

39.4.5.71 status_t SAI_TransferTxSetFormat (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

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

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>format</i>	Pointer to the SAI audio data format structure.
<i>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.

39.4.5.72 `status_t SAI_TransferRxSetFormat (I2S_Type * base, sai_handle_t * handle, sai_transfer_format_t * format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)`

Deprecated Do not use this function. It has been superseded 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`.

39.4.5.73 `status_t SAI_TransferSendNonBlocking (I2S_Type * base, sai_handle_t * handle, sai_transfer_t * xfer)`

Note

This API returns immediately after the transfer initiates. Call the `SAI_TxGetTransferStatusIRQ` to poll the transfer status and check whether the transfer is finished. If the return status is not `kStatus_SAI_Busy`, the transfer is finished.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the <code>sai_handle_t</code> structure which stores the transfer state.
<i>xfer</i>	Pointer to the <code>sai_transfer_t</code> 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.

39.4.5.74 **status_t SAI_TransferReceiveNonBlocking (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_t * *xfer*)**

Note

This API returns immediately after the transfer initiates. Call the SAI_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not *kStatus_SAI_Busy*, the transfer is finished.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	Pointer to the <i>sai_handle_t</i> structure which stores the transfer state.
<i>xfer</i>	Pointer to the 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.

39.4.5.75 **status_t SAI_TransferGetSendCount (I2S_Type * *base*, sai_handle_t * *handle*, size_t * *count*)**

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the <i>sai_handle_t</i> structure which stores the transfer state.
<i>count</i>	Bytes count sent.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is not a non-blocking transaction currently in progress.

39.4.5.76 **status_t SAI_TransferGetReceiveCount (I2S_Type * *base*, sai_handle_t * *handle*, size_t * *count*)**

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.
<i>count</i>	Bytes count received.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is not a non-blocking transaction currently in progress.

39.4.5.77 **void SAI_TransferAbortSend (I2S_Type * *base*, sai_handle_t * *handle*)**

Note

This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.

39.4.5.78 **void SAI_TransferAbortReceive (I2S_Type * *base*, sai_handle_t * *handle*)**

Note

This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.

39.4.5.79 void SAI_TransferTerminateSend (I2S_Type * *base*, sai_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSend.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

39.4.5.80 void SAI_TransferTerminateReceive (I2S_Type * *base*, sai_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceive.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

39.4.5.81 void SAI_TransferTxHandleIRQ (I2S_Type * *base*, sai_handle_t * *handle*)

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure.

39.4.5.82 void SAI_TransferRxHandleIRQ (I2S_Type * *base*, sai_handle_t * *handle*)

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure.

39.5 SAI EDMA Driver

39.5.1 Overview

Data Structures

- struct [sai_edma_handle_t](#)
SAI DMA transfer handle, users should not touch the content of the handle. [More...](#)

Typedefs

- typedef void(* [sai_edma_callback_t](#))(I2S_Type *base, sai_edma_handle_t *handle, [status_t](#) status, void *userData)
SAI eDMA transfer callback function for finish and error.

Driver version

- #define [FSL_SAI_EDMA_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 5, 1))
Version 2.5.1.

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)

- Performs a non-blocking SAI receive using eDMA.*

 - [status_t SAI_TransferSendLoopEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle, [sai_transfer_t](#) *xfer, uint32_t loopTransferCount)
- Performs a non-blocking SAI loop transfer using eDMA.*

 - [status_t SAI_TransferReceiveLoopEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle, [sai_transfer_t](#) *xfer, uint32_t loopTransferCount)
- Performs a non-blocking SAI loop transfer using eDMA.*

 - void [SAI_TransferTerminateSendEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle)
- Terminate all SAI send.*

 - void [SAI_TransferTerminateReceiveEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle)
- Terminate all SAI receive.*

 - void [SAI_TransferAbortSendEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle)
- Aborts a SAI transfer using eDMA.*

 - void [SAI_TransferAbortReceiveEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle)
- Aborts a SAI receive using eDMA.*

 - [status_t SAI_TransferGetSendCountEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle, size_t *count)
- Gets byte count sent by SAI.*

 - [status_t SAI_TransferGetReceiveCountEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle, size_t *count)
- Gets byte count received by SAI.*

 - uint32_t [SAI_TransferGetValidTransferSlotsEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle)
- Gets valid transfer slot.*

39.5.2 Data Structure Documentation

39.5.2.1 struct sai_edma_handle

Data Fields

- [edma_handle_t](#) * [dmaHandle](#)
DMA handler for SAI send.
- uint8_t [nbytes](#)
eDMA minor byte transfer count initially configured.
- uint8_t [bytesPerFrame](#)
Bytes in a frame.
- uint8_t [channelMask](#)
Enabled channel mask value, reference [_sai_channel_mask](#).
- uint8_t [channelNums](#)
total enabled channel nums
- uint8_t [channel](#)
Which data channel.
- uint8_t [count](#)
The transfer data count in a DMA request.
- uint32_t [state](#)
Internal state for SAI eDMA transfer.
- [sai_edma_callback_t](#) [callback](#)
Callback for users while transfer finish or error occurs.
- void * [userData](#)

- *User callback parameter.*
uint8_t `tcd` [(SAI_XFER_QUEUE_SIZE+1U)*sizeof(edma_tcd_t)]
TCD pool for eDMA transfer.
- sai_transfer_t `saiQueue` [SAI_XFER_QUEUE_SIZE]
Transfer queue storing queued transfer.
- size_t `transferSize` [SAI_XFER_QUEUE_SIZE]
Data bytes need to transfer.
- volatile uint8_t `queueUser`
Index for user to queue transfer.
- volatile uint8_t `queueDriver`
Index for driver to get the transfer data and size.

Field Documentation

- (1) uint8_t `sai_edma_handle_t::nbytes`
- (2) uint8_t `sai_edma_handle_t::tcd`[(SAI_XFER_QUEUE_SIZE+1U)*sizeof(edma_tcd_t)]
- (3) sai_transfer_t `sai_edma_handle_t::saiQueue`[SAI_XFER_QUEUE_SIZE]
- (4) volatile uint8_t `sai_edma_handle_t::queueUser`

39.5.3 Function Documentation

39.5.3.1 void SAI_TransferTxCreateHandleEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_edma_callback_t *callback*, void * *userData*, edma_handle_t * *txDmaHandle*)

This function initializes the SAI master DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>base</i>	SAI peripheral base address.
<i>callback</i>	Pointer to user callback function.
<i>userData</i>	User parameter passed to the callback function.
<i>txDmaHandle</i>	eDMA handle pointer, this handle shall be static allocated by users.

39.5.3.2 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.

39.5.3.3 void SAI_TransferTxSetFormatEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

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

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>format</i>	Pointer to SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If bit clock source is master clock, this value should equals to masterClockHz in format.

Return values

<i>kStatus_Success</i>	Audio format set successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

39.5.3.4 void SAI_TransferRxSetFormatEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

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

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>format</i>	Pointer to SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If a bit clock source is the master clock, this value should equal to masterClockHz in format.

Return values

<i>kStatus_Success</i>	Audio format set successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

39.5.3.5 void SAI_TransferTxSetConfigEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transceiver_t * *saiConfig*)

Note

SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of [sai_transceiver_t](#) with the corresponding values of `_sai_channel_mask` enum, enable the FIFO Combine mode by assigning `kSAI_FifoCombineModeEnabledOnWrite` to the `fifoCombine` member of `sai_fifo_combine_t` which is a member of [sai_transceiver_t](#). This is an example of multi-channel data transfer configuration step.

```
*  sai_transceiver_t config;
*  SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits,
*    kSAI_Stereo, kSAI_Channel0Mask|kSAI_Channel1Mask);
*  config.fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnWrite
*    ;
*  SAI_TransferTxSetConfigEDMA(I2S0, &edmaHandle, &config);
*
```

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>saiConfig</i>	sai configurations.

39.5.3.6 void SAI_TransferRxSetConfigEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transceiver_t * *saiConfig*)

Note

SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of *sai_transceiver_t* with the corresponding values of *_sai_channel_mask* enum, enable the FIFO Combine mode by assigning *kSAI_FifoCombineModeEnabledOnRead* to the *fifoCombine* member of *sai_fifo_combine_t* which is a member of *sai_transceiver_t*. This is an example of multi-channel data transfer configuration step.

```
* sai_transceiver_t config;
* SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits,
*   kSAI_Stereo, kSAI_Channel0Mask|kSAI_Channel1Mask);
* config.fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnRead
*   ;
* SAI_TransferRxSetConfigEDMA(I2S0, &edmaHandle, &config);
*
```

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>saiConfig</i>	sai configurations.

39.5.3.7 status_t SAI_TransferSendEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_t * *xfer*)

Note

This interface returns immediately after the transfer initiates. Call *SAI_GetTransferStatus* to poll the transfer status and check whether the SAI transfer is finished.

This function support multi channel transfer,

1. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers
2. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>xfer</i>	Pointer to the DMA transfer structure.

Return values

<i>kStatus_Success</i>	Start a SAI eDMA send successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.
<i>kStatus_TxBusy</i>	SAI is busy sending data.

39.5.3.8 status_t SAI_TransferReceiveEDMA (I2S_Type * base, sai_edma_handle_t * handle, sai_transfer_t * xfer)

Note

This interface returns immediately after the transfer initiates. Call the SAI_GetReceiveRemainingBytes to poll the transfer status and check whether the SAI transfer is finished.

This function support multi channel transfer,

1. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers
2. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI eDMA handle pointer.
<i>xfer</i>	Pointer to DMA transfer structure.

Return values

<i>kStatus_Success</i>	Start a SAI eDMA receive successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

<i>kStatus_RxBusy</i>	SAI is busy receiving data.
-----------------------	-----------------------------

39.5.3.9 **status_t SAI_TransferSendLoopEDMA (I2S_Type * base, sai_edma_handle_t * handle, sai_transfer_t * xfer, uint32_t loopTransferCount)**

Note

This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size. This function support one sai channel only.

Once the loop transfer start, application can use function SAI_TransferAbortSendEDMA to stop the loop transfer.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>xfer</i>	Pointer to the DMA transfer structure, should be a array with elements counts ≥ 1 (loopTransferCount).
<i>loopTransfer-Count</i>	the counts of xfer array.

Return values

<i>kStatus_Success</i>	Start a SAI eDMA send successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

39.5.3.10 **status_t SAI_TransferReceiveLoopEDMA (I2S_Type * base, sai_edma_handle_t * handle, sai_transfer_t * xfer, uint32_t loopTransferCount)**

Note

This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size. This function support one sai channel only.

Once the loop transfer start, application can use function SAI_TransferAbortReceiveEDMA to stop the loop transfer.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>xfer</i>	Pointer to the DMA transfer structure, should be a array with elements counts ≥ 1 (loopTransferCount).
<i>loopTransfer-Count</i>	the counts of xfer array.

Return values

<i>kStatus_Success</i>	Start a SAI eDMA receive successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

39.5.3.11 void SAI_TransferTerminateSendEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSendEDMA.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

39.5.3.12 void SAI_TransferTerminateReceiveEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceiveEDMA.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

39.5.3.13 void SAI_TransferAbortSendEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateSendEDMA.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

39.5.3.14 void SAI_TransferAbortReceiveEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateReceiveEDMA.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI eDMA handle pointer.

39.5.3.15 status_t SAI_TransferGetSendCountEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, size_t * *count*)

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>count</i>	Bytes count sent by SAI.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is no non-blocking transaction in progress.

39.5.3.16 status_t SAI_TransferGetReceiveCountEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, size_t * *count*)

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI eDMA handle pointer.
<i>count</i>	Bytes count received by SAI.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is no non-blocking transaction in progress.

39.5.3.17 `uint32_t SAI_TransferGetValidTransferSlotsEDMA (I2S_Type * base, sai_edma_handle_t * handle)`

This function can be used to query the valid transfer request slot that the application can submit. It should be called in the critical section, that means the application could call it in the corresponding callback function or disable IRQ before calling it in the application, otherwise, the returned value may not correct.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI eDMA handle pointer.

Return values

<i>valid</i>	slot count that application submit.
--------------	-------------------------------------

Chapter 40

SEMC: Smart External DRAM Controller Driver

40.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Smart External DRAM Controller block of MCUXpresso SDK devices.

40.2 SEMC: Smart External DRAM Controller Driver

40.2.1 SEMC Initialization Operation

The SEMC Initialize is to initialize for common configure: gate the SEMC clock, configure IOMUX, and queue weight setting. The SEMC Deinitialize is to ungate the clock and disable SEMC module.

40.2.2 SEMC Interrupt Operation

The interrupt and disable operation for SEMC.

40.2.3 SEMC Memory access Operation

This group is mainly provide NAND/NOR memory access API which is through IP bus/ IP command access. Since the AXI access is directly read/write is so easy, so the AXI read/write part is not provided in SEMC.

40.3 Typical use case

Data Structures

- struct [semc_sdram_config_t](#)
SEMC SDRAM configuration structure. [More...](#)
- struct [semc_nand_timing_config_t](#)
SEMC NAND device timing configuration structure. [More...](#)
- struct [semc_nand_config_t](#)
SEMC NAND configuration structure. [More...](#)
- struct [semc_nor_config_t](#)
SEMC NOR configuration structure. [More...](#)
- struct [semc_sram_config_t](#)
SEMC SRAM configuration structure. [More...](#)
- struct [semc_dbi_config_t](#)
SEMC DBI configuration structure. [More...](#)
- struct [semc_queuea_weight_struct_t](#)

- *SEMC AXI queue a weight setting structure. [More...](#)*
- union `semc_queuea_weight_t`
SEMC AXI queue a weight setting union. [More...](#)
- struct `semc_queueb_weight_struct_t`
SEMC AXI queue b weight setting structure. [More...](#)
- union `semc_queueb_weight_t`
SEMC AXI queue b weight setting union. [More...](#)
- struct `semc_axi_queueweight_t`
SEMC AXI queue weight setting. [More...](#)
- struct `semc_config_t`
SEMC configuration structure. [More...](#)

Enumerations

- enum {
`kStatus_SEMC_InvalidDeviceType = MAKE_STATUS(kStatusGroup_SEMC, 0),`
`kStatus_SEMC_IpCommandExecutionError = MAKE_STATUS(kStatusGroup_SEMC, 1),`
`kStatus_SEMC_AxiCommandExecutionError = MAKE_STATUS(kStatusGroup_SEMC, 2),`
`kStatus_SEMC_InvalidMemorySize = MAKE_STATUS(kStatusGroup_SEMC, 3),`
`kStatus_SEMC_InvalidIpcmdDataSize = MAKE_STATUS(kStatusGroup_SEMC, 4),`
`kStatus_SEMC_InvalidAddressPortWidth = MAKE_STATUS(kStatusGroup_SEMC, 5),`
`kStatus_SEMC_InvalidDataPortWidth = MAKE_STATUS(kStatusGroup_SEMC, 6),`
`kStatus_SEMC_InvalidSwPinmuxSelection = MAKE_STATUS(kStatusGroup_SEMC, 7),`
`kStatus_SEMC_InvalidBurstLength = MAKE_STATUS(kStatusGroup_SEMC, 8),`
`kStatus_SEMC_InvalidColumnAddressBitWidth = MAKE_STATUS(kStatusGroup_SEMC, 9),`
`kStatus_SEMC_InvalidBaseAddress = MAKE_STATUS(kStatusGroup_SEMC, 10),`
`kStatus_SEMC_InvalidTimerSetting = MAKE_STATUS(kStatusGroup_SEMC, 11) }`
SEMC status, `_semc_status`.
- enum `semc_mem_type_t` {
`kSEMC_MemType_SDRAM = 0,`
`kSEMC_MemType_SRAM,`
`kSEMC_MemType_NOR,`
`kSEMC_MemType_NAND,`
`kSEMC_MemType_8080 }`
SEMC memory device type.
- enum `semc_waitready_polarity_t` {
`kSEMC_LowActive = 0,`
`kSEMC_HighActive }`
SEMC WAIT/RDY polarity.
- enum `semc_sdram_cs_t` {
`kSEMC_SDRAM_CS0 = 0,`
`kSEMC_SDRAM_CS1,`
`kSEMC_SDRAM_CS2,`
`kSEMC_SDRAM_CS3 }`
SEMC SDRAM Chip selection .
- enum `semc_sram_cs_t` { `kSEMC_SRAM_CS0 = 0 }`
SEMC SRAM Chip selection .
- enum `semc_nand_access_type_t` {

```
kSEMC_NAND_ACCESS_BY_AXI = 0,
kSEMC_NAND_ACCESS_BY_IPCMD }
```

SEMC NAND device type.

- enum `semc_interrupt_enable_t` {


```
kSEMC_IPCmdDoneInterrupt = SEMC_INTEN_IPCMDDONEEN_MASK,
kSEMC_IPCmdErrInterrupt = SEMC_INTEN_IPCMDERREN_MASK,
kSEMC_AXICmdErrInterrupt = SEMC_INTEN_AXICMDERREN_MASK,
kSEMC_AXIBusErrInterrupt = SEMC_INTEN_AXIBUSERREN_MASK }
```

SEMC interrupts .

- enum `semc_ipcmd_datasize_t` {


```
kSEMC_IPcmdDataSize_1bytes = 1,
kSEMC_IPcmdDataSize_2bytes,
kSEMC_IPcmdDataSize_3bytes,
kSEMC_IPcmdDataSize_4bytes }
```

SEMC IP command data size in bytes.

- enum `semc_refresh_time_t` {


```
kSEMC_RefreshThreeClocks = 0x0U,
kSEMC_RefreshSixClocks,
kSEMC_RefreshNineClocks }
```

SEMC auto-refresh timing.

- enum `semc_caslatency_t` {


```
kSEMC_LatencyOne = 1,
kSEMC_LatencyTwo,
kSEMC_LatencyThree }
```

CAS latency.

- enum `semc_sdram_column_bit_num_t` {


```
kSEMC_SdramColumn_12bit = 0x0U,
kSEMC_SdramColumn_11bit,
kSEMC_SdramColumn_10bit,
kSEMC_SdramColumn_9bit }
```

SEMC sdram column address bit number.

- enum `sem_sdram_burst_len_t` {


```
kSEMC_Sdram_BurstLen1 = 0,
kSEMC_Sdram_BurstLen2,
kSEMC_Sdram_BurstLen4,
kSEMC_Sdram_BurstLen8 }
```

SEMC sdram burst length.

- enum `semc_nand_column_bit_num_t` {


```
kSEMC_NandColum_16bit = 0x0U,
kSEMC_NandColum_15bit,
kSEMC_NandColum_14bit,
kSEMC_NandColum_13bit,
kSEMC_NandColum_12bit,
kSEMC_NandColum_11bit,
kSEMC_NandColum_10bit,
kSEMC_NandColum_9bit }
```

- *SEMC nand column address bit number.*

```
enum sem_nand_burst_len_t {
    kSEMC_Nand_BurstLen1 = 0,
    kSEMC_Nand_BurstLen2,
    kSEMC_Nand_BurstLen4,
    kSEMC_Nand_BurstLen8,
    kSEMC_Nand_BurstLen16,
    kSEMC_Nand_BurstLen32,
    kSEMC_Nand_BurstLen64 }
    SEMC nand burst length.
```
- ```
enum semc_norsram_column_bit_num_t {
 kSEMC_NorColum_12bit = 0x0U,
 kSEMC_NorColum_11bit,
 kSEMC_NorColum_10bit,
 kSEMC_NorColum_9bit,
 kSEMC_NorColum_8bit,
 kSEMC_NorColum_7bit,
 kSEMC_NorColum_6bit,
 kSEMC_NorColum_5bit,
 kSEMC_NorColum_4bit,
 kSEMC_NorColum_3bit,
 kSEMC_NorColum_2bit }
 SEMC nor/sram column address bit number.
```
- ```
enum sem_norsram_burst_len_t {
    kSEMC_Nor_BurstLen1 = 0,
    kSEMC_Nor_BurstLen2,
    kSEMC_Nor_BurstLen4,
    kSEMC_Nor_BurstLen8,
    kSEMC_Nor_BurstLen16,
    kSEMC_Nor_BurstLen32,
    kSEMC_Nor_BurstLen64 }
    SEMC nor/sram burst length.
```
- ```
enum semc_dbi_column_bit_num_t {
 kSEMC_Dbi_Colum_12bit = 0x0U,
 kSEMC_Dbi_Colum_11bit,
 kSEMC_Dbi_Colum_10bit,
 kSEMC_Dbi_Colum_9bit,
 kSEMC_Dbi_Colum_8bit,
 kSEMC_Dbi_Colum_7bit,
 kSEMC_Dbi_Colum_6bit,
 kSEMC_Dbi_Colum_5bit,
 kSEMC_Dbi_Colum_4bit,
 kSEMC_Dbi_Colum_3bit,
 kSEMC_Dbi_Colum_2bit }
 SEMC dbi column address bit number.
```
- ```
enum sem_dbi_burst_len_t {
```


- ```

kSEMC_Dbi_BurstLen1 = 0,
kSEMC_Dbi_BurstLen2,
kSEMC_Dbi_Dbi_BurstLen4,
kSEMC_Dbi_BurstLen8,
kSEMC_Dbi_BurstLen16,
kSEMC_Dbi_BurstLen32,
kSEMC_Dbi_BurstLen64 }
 SEMC dbi burst length.

```
- enum `semc_iomux_pin` {

```

kSEMC_MUXA8 = SEMC_IOC_R_MUX_A8_SHIFT,
kSEMC_MUXCSX0 = SEMC_IOC_R_MUX_CSX0_SHIFT,
kSEMC_MUXCSX1 = SEMC_IOC_R_MUX_CSX1_SHIFT,
kSEMC_MUXCSX2 = SEMC_IOC_R_MUX_CSX2_SHIFT,
kSEMC_MUXCSX3 = SEMC_IOC_R_MUX_CSX3_SHIFT,
kSEMC_MUXRDY = SEMC_IOC_R_MUX_RDY_SHIFT }
 SEMC IOMUXC.

```
  - enum `semc_iomux_nora27_pin` {

```

kSEMC_MORA27_NONE = 0,
kSEMC_NORA27_MUXCSX3 = SEMC_IOC_R_MUX_CSX3_SHIFT,
kSEMC_NORA27_MUXRDY = SEMC_IOC_R_MUX_RDY_SHIFT }
 SEMC NOR/PSRAM Address bit 27 A27.

```
  - enum `semc_port_size_t` {

```

kSEMC_PortSize8Bit = 0,
kSEMC_PortSize16Bit }
 SEMC port size.

```
  - enum `semc_addr_mode_t` {

```

kSEMC_AddrDataMux = 0,
kSEMC_AdvAddrdataMux,
kSEMC_AddrDataNonMux }
 SEMC address mode.

```
  - enum `semc_dqs_mode_t` {

```

kSEMC_Loopbackinternal = 0,
kSEMC_Loopbackdqspad }
 SEMC DQS read strobe mode.

```
  - enum `semc_adv_polarity_t` {

```

kSEMC_AdvActiveLow = 0,
kSEMC_AdvActiveHigh }
 SEMC ADV signal active polarity.

```
  - enum `semc_sync_mode_t` {

```

kSEMC_AsyncMode = 0,
kSEMC_SyncMode }
 SEMC sync mode.

```
  - enum `semc_adv_level_control_t` {

```

kSEMC_AdvHigh = 0,
kSEMC_AdvLow }
 SEMC ADV signal level control.

```
  - enum `semc_rdy_polarity_t` {

```
kSEMC_RdyActiveLow = 0,
kSEMC_RdyActivehigh }
```

*SEMC RDY signal active polarity.*

- enum `semc_ipcmd_nand_addrmode_t` {
 

```
kSEMC_NANDAM_ColumnRow = 0x0U,
kSEMC_NANDAM_ColumnCA0,
kSEMC_NANDAM_ColumnCA0CA1,
kSEMC_NANDAM_RawRA0,
kSEMC_NANDAM_RawRA0RA1,
kSEMC_NANDAM_RawRA0RA1RA2 }
```

*SEMC IP command for NAND: address mode.*
- enum `semc_ipcmd_nand_cmdmode_t` {
 

```
kSEMC_NANDCM_Command = 0x2U,
kSEMC_NANDCM_CommandHold,
kSEMC_NANDCM_CommandAddress,
kSEMC_NANDCM_CommandAddressHold,
kSEMC_NANDCM_CommandAddressRead,
kSEMC_NANDCM_CommandAddressWrite,
kSEMC_NANDCM_CommandRead,
kSEMC_NANDCM_CommandWrite,
kSEMC_NANDCM_Read,
kSEMC_NANDCM_Write }
```

*SEMC IP command for NAND command mode.*
- enum `semc_nand_address_option_t` {
 

```
kSEMC_NandAddrOption_5byte_CA2RA3 = 0U,
kSEMC_NandAddrOption_4byte_CA2RA2 = 2U,
kSEMC_NandAddrOption_3byte_CA2RA1 = 4U,
kSEMC_NandAddrOption_4byte_CA1RA3 = 1U,
kSEMC_NandAddrOption_3byte_CA1RA2 = 3U,
kSEMC_NandAddrOption_2byte_CA1RA1 = 7U }
```

*SEMC NAND address option.*
- enum `semc_ipcmd_nor_dbi_t` {
 

```
kSEMC_NORDBICM_Read = 0x2U,
kSEMC_NORDBICM_Write }
```

*SEMC IP command for NOR.*
- enum `semc_ipcmd_sram_t` {
 

```
kSEMC_SRAMCM_ArrayRead = 0x2U,
kSEMC_SRAMCM_ArrayWrite,
kSEMC_SRAMCM_RegRead,
kSEMC_SRAMCM_RegWrite }
```

*SEMC IP command for SRAM.*
- enum `semc_ipcmd_sdram_t` {

```

kSEMC_SDRAMCM_Read = 0x8U,
kSEMC_SDRAMCM_Write,
kSEMC_SDRAMCM_Modeset,
kSEMC_SDRAMCM_Active,
kSEMC_SDRAMCM_AutoRefresh,
kSEMC_SDRAMCM_SelfRefresh,
kSEMC_SDRAMCM_Precharge,
kSEMC_SDRAMCM_Prechargeall }
SEMC IP command for SDARM.

```

## Driver version

- #define `FSL_SEMC_DRIVER_VERSION` (`MAKE_VERSION(2, 6, 0)`)  
*SEMC driver version.*

## SEMC Initialization and De-initialization

- void `SEMC_GetDefaultConfig` (`semc_config_t *config`)  
*Gets the SEMC default basic configuration structure.*
- void `SEMC_Init` (`SEMC_Type *base, semc_config_t *configure`)  
*Initializes SEMC.*
- void `SEMC_Deinit` (`SEMC_Type *base`)  
*Deinitializes the SEMC module and gates the clock.*

## SEMC Configuration Operation For Each Memory Type

- `status_t SEMC_ConfigureSDRAM` (`SEMC_Type *base, semc_sdram_cs_t cs, semc_sdram_config_t *config, uint32_t clkSrc_Hz`)  
*Configures SDRAM controller in SEMC.*
- `status_t SEMC_ConfigureNAND` (`SEMC_Type *base, semc_nand_config_t *config, uint32_t clkSrc_Hz`)  
*Configures NAND controller in SEMC.*
- `status_t SEMC_ConfigureNOR` (`SEMC_Type *base, semc_nor_config_t *config, uint32_t clkSrc_Hz`)  
*Configures NOR controller in SEMC.*
- `status_t SEMC_ConfigureSRAMWithChipSelection` (`SEMC_Type *base, semc_sram_cs_t cs, semc_sram_config_t *config, uint32_t clkSrc_Hz`)  
*Configures SRAM controller in SEMC.*
- `status_t SEMC_ConfigureSRAM` (`SEMC_Type *base, semc_sram_config_t *config, uint32_t clkSrc_Hz`)  
*Configures SRAM controller in SEMC.*
- `status_t SEMC_ConfigureDBI` (`SEMC_Type *base, semc_dbi_config_t *config, uint32_t clkSrc_Hz`)  
*Configures DBI controller in SEMC.*

## SEMC Interrupt Operation

- static void `SEMC_EnableInterrupts` (`SEMC_Type *base, uint32_t mask`)  
*Enables the SEMC interrupt.*

- static void [SEMC\\_DisableInterrupts](#) (SEMC\_Type \*base, uint32\_t mask)  
*Disables the SEMC interrupt.*
- static bool [SEMC\\_GetStatusFlag](#) (SEMC\_Type \*base)  
*Gets the SEMC status.*
- static void [SEMC\\_ClearStatusFlags](#) (SEMC\_Type \*base, uint32\_t mask)  
*Clears the SEMC status flag state.*

## SEMC Memory Access Operation

- static bool [SEMC\\_IsInIdle](#) (SEMC\_Type \*base)  
*Check if SEMC is in idle.*
- [status\\_t SEMC\\_SendIPCommand](#) (SEMC\_Type \*base, [semc\\_mem\\_type\\_t](#) memType, uint32\_t address, uint32\_t command, uint32\_t write, uint32\_t \*read)  
*SEMC IP command access.*
- static uint16\_t [SEMC\\_BuildNandIPCommand](#) (uint8\_t userCommand, [semc\\_ipcmd\\_nand\\_addrmode\\_t](#) addrMode, [semc\\_ipcmd\\_nand\\_cmdmode\\_t](#) cmdMode)  
*Build SEMC IP command for NAND.*
- static bool [SEMC\\_IsNandReady](#) (SEMC\_Type \*base)  
*Check if the NAND device is ready.*
- [status\\_t SEMC\\_IPCommandNandWrite](#) (SEMC\_Type \*base, uint32\_t address, uint8\_t \*data, uint32\_t size\_bytes)  
*SEMC NAND device memory write through IP command.*
- [status\\_t SEMC\\_IPCommandNandRead](#) (SEMC\_Type \*base, uint32\_t address, uint8\_t \*data, uint32\_t size\_bytes)  
*SEMC NAND device memory read through IP command.*
- [status\\_t SEMC\\_IPCommandNorWrite](#) (SEMC\_Type \*base, uint32\_t address, uint8\_t \*data, uint32\_t size\_bytes)  
*SEMC NOR device memory write through IP command.*
- [status\\_t SEMC\\_IPCommandNorRead](#) (SEMC\_Type \*base, uint32\_t address, uint8\_t \*data, uint32\_t size\_bytes)  
*SEMC NOR device memory read through IP command.*

## 40.4 Data Structure Documentation

### 40.4.1 struct [semc\\_sdram\\_config\\_t](#)

1. The memory size in the configuration is in the unit of KB. So memsize\_kbytes should be set as  $2^2$ ,  $2^3$ ,  $2^4$  .etc which is base 2KB exponential function. Take refer to BR0~BR3 register in RM for details.
2. The prescalePeriod\_N16Cycle is in unit of 16 clock cycle. It is a exception for prescaleTimer\_n16cycle = 0, it means the prescaler timer period is  $256 * 16$  clock cycles. For precalerIf precalerTimer\_n16cycle not equal to 0, The prescaler timer period is prescalePeriod\_N16Cycle \* 16 clock cycles. idleTimeout\_NprescalePeriod, refreshUrgThreshold\_NprescalePeriod, refreshPeriod\_NprescalePeriod are similar to prescalePeriod\_N16Cycle.

### Data Fields

- [semc\\_iomux\\_pin csxPinMux](#)

- *CS pin mux.*
- `uint32_t address`  
*The base address.*
- `uint32_t memsize_kbytes`  
*The memory size in unit of kbytes.*
- `smec_port_size_t portSize`  
*Port size.*
- `sem_sdram_burst_len_t burstLen`  
*Burst length.*
- `semc_sdram_column_bit_num_t columnAddrBitNum`  
*Column address bit number.*
- `semc_caslatency_t casLatency`  
*CAS latency.*
- `uint8_t tPrecharge2Act_Ns`  
*Precharge to active wait time in unit of nanosecond.*
- `uint8_t tAct2ReadWrite_Ns`  
*Act to read/write wait time in unit of nanosecond.*
- `uint8_t tRefreshRecovery_Ns`  
*Refresh recovery time in unit of nanosecond.*
- `uint8_t tWriteRecovery_Ns`  
*write recovery time in unit of nanosecond.*
- `uint8_t tCkeOff_Ns`  
*CKE off minimum time in unit of nanosecond.*
- `uint8_t tAct2Prechage_Ns`  
*Active to precharge in unit of nanosecond.*
- `uint8_t tSelfRefRecovery_Ns`  
*Self refresh recovery time in unit of nanosecond.*
- `uint8_t tRefresh2Refresh_Ns`  
*Refresh to refresh wait time in unit of nanosecond.*
- `uint8_t tAct2Act_Ns`  
*Active to active wait time in unit of nanosecond.*
- `uint32_t tPrescalePeriod_Ns`  
*Prescaler timer period should not be larger than  $256 * 16 * \text{clock cycle}$ .*
- `uint32_t tIdleTimeout_Ns`  
*Idle timeout in unit of prescale time period.*
- `uint32_t refreshPeriod_nsPerRow`  
*Refresh timer period like  $64\text{ms} * 1000000/8192$ .*
- `uint32_t refreshUrgThreshold`  
*Refresh urgent threshold.*
- `uint8_t refreshBurstLen`  
*Refresh burst length.*

## Field Documentation

### (1) `semc_iomux_pin semc_sdram_config_t::csxPinMux`

The `kSEMC_MUXA8` is not valid in sdram pin mux setting.

- (2) `uint32_t semc_sdram_config_t::address`
- (3) `uint32_t semc_sdram_config_t::memsize_kbytes`
- (4) `smec_port_size_t semc_sdram_config_t::portSize`
- (5) `sem_sdram_burst_len_t semc_sdram_config_t::burstLen`
- (6) `semc_sdram_column_bit_num_t semc_sdram_config_t::columnAddrBitNum`
- (7) `semc_caslatency_t semc_sdram_config_t::casLatency`
- (8) `uint8_t semc_sdram_config_t::tPrecharge2Act_Ns`
- (9) `uint8_t semc_sdram_config_t::tAct2ReadWrite_Ns`
- (10) `uint8_t semc_sdram_config_t::tRefreshRecovery_Ns`
- (11) `uint8_t semc_sdram_config_t::tWriteRecovery_Ns`
- (12) `uint8_t semc_sdram_config_t::tCkeOff_Ns`
- (13) `uint8_t semc_sdram_config_t::tAct2Prechage_Ns`
- (14) `uint8_t semc_sdram_config_t::tSelfRefRecovery_Ns`
- (15) `uint8_t semc_sdram_config_t::tRefresh2Refresh_Ns`
- (16) `uint8_t semc_sdram_config_t::tAct2Act_Ns`
- (17) `uint32_t semc_sdram_config_t::tPrescalePeriod_Ns`
- (18) `uint32_t semc_sdram_config_t::tIdleTimeout_Ns`
- (19) `uint32_t semc_sdram_config_t::refreshPeriod_nsPerRow`
- (20) `uint32_t semc_sdram_config_t::refreshUrgThreshold`
- (21) `uint8_t semc_sdram_config_t::refreshBurstLen`

#### 40.4.2 struct `semc_nand_timing_config_t`

##### Data Fields

- `uint8_t tCeSetup_Ns`  
*CE setup time: tCS.*
- `uint8_t tCeHold_Ns`  
*CE hold time: tCH.*
- `uint8_t tCeInterval_Ns`  
*CE interval time: tCEITV.*

- uint8\_t [tWeLow\\_Ns](#)  
*WE low time: tWP.*
- uint8\_t [tWeHigh\\_Ns](#)  
*WE high time: tWH.*
- uint8\_t [tReLow\\_Ns](#)  
*RE low time: tRP.*
- uint8\_t [tReHigh\\_Ns](#)  
*RE high time: tREH.*
- uint8\_t [tTurnAround\\_Ns](#)  
*Turnaround time for async mode: tTA.*
- uint8\_t [tWehigh2Relow\\_Ns](#)  
*WE# high to RE# wait time: tWHR.*
- uint8\_t [tRehigh2Welow\\_Ns](#)  
*RE# high to WE# low wait time: tRHW.*
- uint8\_t [tAle2WriteStart\\_Ns](#)  
*ALE to write start wait time: tADL.*
- uint8\_t [tReady2Relow\\_Ns](#)  
*Ready to RE# low min wait time: tRR.*
- uint8\_t [tWehigh2Busy\\_Ns](#)  
*WE# high to busy wait time: tWB.*

## Field Documentation

- (1) `uint8_t semc_nand_timing_config_t::tCeSetup_Ns`
- (2) `uint8_t semc_nand_timing_config_t::tCeHold_Ns`
- (3) `uint8_t semc_nand_timing_config_t::tCeInterval_Ns`
- (4) `uint8_t semc_nand_timing_config_t::tWeLow_Ns`
- (5) `uint8_t semc_nand_timing_config_t::tWeHigh_Ns`
- (6) `uint8_t semc_nand_timing_config_t::tReLow_Ns`
- (7) `uint8_t semc_nand_timing_config_t::tReHigh_Ns`
- (8) `uint8_t semc_nand_timing_config_t::tTurnAround_Ns`
- (9) `uint8_t semc_nand_timing_config_t::tWehigh2Relow_Ns`
- (10) `uint8_t semc_nand_timing_config_t::tRehigh2Welow_Ns`
- (11) `uint8_t semc_nand_timing_config_t::tAle2WriteStart_Ns`
- (12) `uint8_t semc_nand_timing_config_t::tReady2Relow_Ns`
- (13) `uint8_t semc_nand_timing_config_t::tWehigh2Busy_Ns`

### 40.4.3 struct `semc_nand_config_t`

#### Data Fields

- `semc_iomux_pin cePinMux`  
*The CE pin mux setting.*
- `uint32_t axiAddress`  
*The base address for AXI nand.*
- `uint32_t axiMemsize_kbytes`  
*The memory size in unit of kbytes for AXI nand.*
- `uint32_t ipgAddress`  
*The base address for IPG nand .*
- `uint32_t ipgMemsize_kbytes`  
*The memory size in unit of kbytes for IPG nand.*
- `semc_rdy_polarity_t rdyactivePolarity`  
*Wait ready polarity.*
- `bool edoModeEnabled`  
*EDO mode enabled.*
- `semc_nand_column_bit_num_t columnAddrBitNum`  
*Column address bit number.*
- `semc_nand_address_option_t arrayAddrOption`  
*Address option.*



- [sem\\_nand\\_burst\\_len\\_t burstLen](#)  
*Burst length.*
- [smec\\_port\\_size\\_t portSize](#)  
*Port size.*
- [semc\\_nand\\_timing\\_config\\_t \\* timingConfig](#)  
*SEMC nand timing configuration.*

### Field Documentation

#### (1) [semc\\_iomux\\_pin semc\\_nand\\_config\\_t::cePinMux](#)

The kSEMC\_MUXRDY is not valid for CE pin setting.

#### (2) [uint32\\_t semc\\_nand\\_config\\_t::axiAddress](#)

#### (3) [uint32\\_t semc\\_nand\\_config\\_t::axiMemsize\\_kbytes](#)

#### (4) [uint32\\_t semc\\_nand\\_config\\_t::ipgAddress](#)

#### (5) [uint32\\_t semc\\_nand\\_config\\_t::ipgMemsize\\_kbytes](#)

#### (6) [semc\\_rdy\\_polarity\\_t semc\\_nand\\_config\\_t::rdyactivePolarity](#)

#### (7) [bool semc\\_nand\\_config\\_t::edoModeEnabled](#)

#### (8) [semc\\_nand\\_column\\_bit\\_num\\_t semc\\_nand\\_config\\_t::columnAddrBitNum](#)

#### (9) [semc\\_nand\\_address\\_option\\_t semc\\_nand\\_config\\_t::arrayAddrOption](#)

#### (10) [sem\\_nand\\_burst\\_len\\_t semc\\_nand\\_config\\_t::burstLen](#)

#### (11) [smec\\_port\\_size\\_t semc\\_nand\\_config\\_t::portSize](#)

#### (12) [semc\\_nand\\_timing\\_config\\_t\\* semc\\_nand\\_config\\_t::timingConfig](#)

### 40.4.4 struct [semc\\_nor\\_config\\_t](#)

#### Data Fields

- [semc\\_iomux\\_pin cePinMux](#)  
*The CE# pin mux setting.*
- [semc\\_iomux\\_nora27\\_pin addr27](#)  
*The Addr bit 27 pin mux setting.*
- [uint32\\_t address](#)  
*The base address.*
- [uint32\\_t memsize\\_kbytes](#)  
*The memory size in unit of kbytes.*
- [uint8\\_t addrPortWidth](#)  
*The address port width.*
- [semc\\_rdy\\_polarity\\_t rdyactivePolarity](#)

- *Wait ready polarity.*
- [semc\\_adv\\_polarity\\_t](#) advActivePolarity  
*ADV# polarity.*
- [semc\\_norsram\\_column\\_bit\\_num\\_t](#) columnAddrBitNum  
*Column address bit number.*
- [semc\\_addr\\_mode\\_t](#) addrMode  
*Address mode.*
- [sem\\_norsram\\_burst\\_len\\_t](#) burstLen  
*Burst length.*
- [smec\\_port\\_size\\_t](#) portSize  
*Port size.*
- [uint8\\_t](#) tCeSetup\_Ns  
*The CE setup time.*
- [uint8\\_t](#) tCeHold\_Ns  
*The CE hold time.*
- [uint8\\_t](#) tCeInterval\_Ns  
*CE interval minimum time.*
- [uint8\\_t](#) tAddrSetup\_Ns  
*The address setup time.*
- [uint8\\_t](#) tAddrHold\_Ns  
*The address hold time.*
- [uint8\\_t](#) tWeLow\_Ns  
*WE low time for async mode.*
- [uint8\\_t](#) tWeHigh\_Ns  
*WE high time for async mode.*
- [uint8\\_t](#) tReLow\_Ns  
*RE low time for async mode.*
- [uint8\\_t](#) tReHigh\_Ns  
*RE high time for async mode.*
- [uint8\\_t](#) tTurnAround\_Ns  
*Turnaround time for async mode.*
- [uint8\\_t](#) tAddr2WriteHold\_Ns  
*Address to write data hold time for async mode.*



## Field Documentation

- (1) `semc_iomux_pin semc_nor_config_t::cePinMux`
- (2) `semc_iomux_nora27_pin semc_nor_config_t::addr27`
- (3) `uint32_t semc_nor_config_t::address`
- (4) `uint32_t semc_nor_config_t::memsize_kbytes`
- (5) `uint8_t semc_nor_config_t::addrPortWidth`
- (6) `semc_rdy_polarity_t semc_nor_config_t::rdyactivePolarity`
- (7) `semc_adv_polarity_t semc_nor_config_t::advActivePolarity`
- (8) `semc_norsram_column_bit_num_t semc_nor_config_t::columnAddrBitNum`
- (9) `semc_addr_mode_t semc_nor_config_t::addrMode`
- (10) `sem_norsram_burst_len_t semc_nor_config_t::burstLen`
- (11) `smec_port_size_t semc_nor_config_t::portSize`
- (12) `uint8_t semc_nor_config_t::tCeSetup_Ns`
- (13) `uint8_t semc_nor_config_t::tCeHold_Ns`
- (14) `uint8_t semc_nor_config_t::tCeInterval_Ns`
- (15) `uint8_t semc_nor_config_t::tAddrSetup_Ns`
- (16) `uint8_t semc_nor_config_t::tAddrHold_Ns`
- (17) `uint8_t semc_nor_config_t::tWeLow_Ns`
- (18) `uint8_t semc_nor_config_t::tWeHigh_Ns`
- (19) `uint8_t semc_nor_config_t::tReLow_Ns`
- (20) `uint8_t semc_nor_config_t::tReHigh_Ns`
- (21) `uint8_t semc_nor_config_t::tTurnAround_Ns`
- (22) `uint8_t semc_nor_config_t::tAddr2WriteHold_Ns`

### 40.4.5 struct `semc_sram_config_t`

#### Data Fields

- [semc\\_iomux\\_pin cePinMux](#)

- *The CE# pin mux setting.*
- [semc\\_iomux\\_nora27\\_pin addr27](#)  
*The Addr bit 27 pin mux setting.*
- [uint32\\_t address](#)  
*The base address.*
- [uint32\\_t memsize\\_kbytes](#)  
*The memory size in unit of kbytes.*
- [uint8\\_t addrPortWidth](#)  
*The address port width.*
- [semc\\_adv\\_polarity\\_t advActivePolarity](#)  
*ADV# polarity 1: active high, 0: active low.*
- [semc\\_addr\\_mode\\_t addrMode](#)  
*Address mode.*
- [sem\\_norsram\\_burst\\_len\\_t burstLen](#)  
*Burst length.*
- [smec\\_port\\_size\\_t portSize](#)  
*Port size.*
- [uint8\\_t tCeSetup\\_Ns](#)  
*The CE setup time.*
- [uint8\\_t tCeHold\\_Ns](#)  
*The CE hold time.*
- [uint8\\_t tCeInterval\\_Ns](#)  
*CE interval minimum time.*
- [uint8\\_t tAddrSetup\\_Ns](#)  
*The address setup time.*
- [uint8\\_t tAddrHold\\_Ns](#)  
*The address hold time.*
- [uint8\\_t tWeLow\\_Ns](#)  
*WE low time for async mode.*
- [uint8\\_t tWeHigh\\_Ns](#)  
*WE high time for async mode.*
- [uint8\\_t tReLow\\_Ns](#)  
*RE low time for async mode.*
- [uint8\\_t tReHigh\\_Ns](#)  
*RE high time for async mode.*
- [uint8\\_t tTurnAround\\_Ns](#)  
*Turnaround time for async mode.*
- [uint8\\_t tAddr2WriteHold\\_Ns](#)  
*Address to write data hold time for async mode.*

## Field Documentation

- (1) `semc_iomux_pin` `semc_sram_config_t::cePinMux`
- (2) `semc_iomux_nora27_pin` `semc_sram_config_t::addr27`
- (3) `uint32_t` `semc_sram_config_t::address`
- (4) `uint32_t` `semc_sram_config_t::memsize_kbytes`
- (5) `uint8_t` `semc_sram_config_t::addrPortWidth`
- (6) `semc_adv_polarity_t` `semc_sram_config_t::advActivePolarity`
- (7) `semc_addr_mode_t` `semc_sram_config_t::addrMode`
- (8) `sem_norsram_burst_len_t` `semc_sram_config_t::burstLen`
- (9) `smec_port_size_t` `semc_sram_config_t::portSize`
- (10) `uint8_t` `semc_sram_config_t::tCeSetup_Ns`
- (11) `uint8_t` `semc_sram_config_t::tCeHold_Ns`
- (12) `uint8_t` `semc_sram_config_t::tCeInterval_Ns`
- (13) `uint8_t` `semc_sram_config_t::tAddrSetup_Ns`
- (14) `uint8_t` `semc_sram_config_t::tAddrHold_Ns`
- (15) `uint8_t` `semc_sram_config_t::tWeLow_Ns`
- (16) `uint8_t` `semc_sram_config_t::tWeHigh_Ns`
- (17) `uint8_t` `semc_sram_config_t::tReLow_Ns`
- (18) `uint8_t` `semc_sram_config_t::tReHigh_Ns`
- (19) `uint8_t` `semc_sram_config_t::tTurnAround_Ns`
- (20) `uint8_t` `semc_sram_config_t::tAddr2WriteHold_Ns`

40.4.6 `struct semc_dbi_config_t`

## Data Fields

- `semc_iomux_pin` `csxPinMux`  
*The CE# pin mux.*
- `uint32_t` `address`  
*The base address.*

- `uint32_t memsize_kbytes`  
*The memory size in unit of 4kbytes.*
- `semc_dbi_column_bit_num_t columnAddrBitNum`  
*Column address bit number.*
- `sem_dbi_burst_len_t burstLen`  
*Burst length.*
- `smec_port_size_t portSize`  
*Port size.*
- `uint8_t tCsxSetup_Ns`  
*The CSX setup time.*
- `uint8_t tCsxHold_Ns`  
*The CSX hold time.*
- `uint8_t tWexLow_Ns`  
*WEX low time.*
- `uint8_t tWexHigh_Ns`  
*WEX high time.*
- `uint8_t tRdxLow_Ns`  
*RDX low time.*
- `uint8_t tRdxHigh_Ns`  
*RDX high time.*
- `uint8_t tCsxInterval_Ns`  
*Write data setup time.*

## Field Documentation

- (1) `semc_iomux_pin semc_dbi_config_t::csxPinMux`
- (2) `uint32_t semc_dbi_config_t::address`
- (3) `uint32_t semc_dbi_config_t::memsize_kbytes`
- (4) `semc_dbi_column_bit_num_t semc_dbi_config_t::columnAddrBitNum`
- (5) `sem_dbi_burst_len_t semc_dbi_config_t::burstLen`
- (6) `smec_port_size_t semc_dbi_config_t::portSize`
- (7) `uint8_t semc_dbi_config_t::tCsxSetup_Ns`
- (8) `uint8_t semc_dbi_config_t::tCsxHold_Ns`
- (9) `uint8_t semc_dbi_config_t::tWexLow_Ns`
- (10) `uint8_t semc_dbi_config_t::tWexHigh_Ns`
- (11) `uint8_t semc_dbi_config_t::tRdxLow_Ns`
- (12) `uint8_t semc_dbi_config_t::tRdxHigh_Ns`
- (13) `uint8_t semc_dbi_config_t::tCsxInterval_Ns`

40.4.7 `struct semc_queuea_weight_struct_t`

## Data Fields

- `uint32_t qos`: 4  
*weight of qos for queue 0.*
- `uint32_t aging`: 4  
*weight of aging for queue 0.*
- `uint32_t slaveHitNoswitch`: 8  
*weight of read/write no switch for queue 0.*
- `uint32_t slaveHitSwitch`: 8  
*weight of read/write switch for queue 0.*



**Field Documentation**

- (1) `uint32_t semc_queuea_weight_struct_t::qos`
- (2) `uint32_t semc_queuea_weight_struct_t::aging`
- (3) `uint32_t semc_queuea_weight_struct_t::slaveHitNoswitch`
- (4) `uint32_t semc_queuea_weight_struct_t::slaveHitSwitch`

**40.4.8 union semc\_queuea\_weight\_t****Data Fields**

- [semc\\_queuea\\_weight\\_struct\\_t queueaConfig](#)  
*Structure configuration for queueA.*
- `uint32_t queueaValue`  
*Configuration value for queueA which could directly write to the reg.*

**Field Documentation**

- (1) `semc_queuea_weight_struct_t semc_queuea_weight_t::queueaConfig`
- (2) `uint32_t semc_queuea_weight_t::queueaValue`

**40.4.9 struct semc\_queueb\_weight\_struct\_t****Data Fields**

- `uint32_t qos`: 4  
*weight of qos for queue 1.*
- `uint32_t aging`: 4  
*weight of aging for queue 1.*
- `uint32_t weightPagehit`: 8  
*weight of page hit for queue 1 only .*
- `uint32_t slaveHitNoswitch`: 8  
*weight of read/write no switch for queue 1.*
- `uint32_t bankRotation`: 8  
*weight of bank rotation for queue 1 only .*

**Field Documentation**

- (1) `uint32_t semc_queueb_weight_struct_t::qos`
- (2) `uint32_t semc_queueb_weight_struct_t::aging`
- (3) `uint32_t semc_queueb_weight_struct_t::weightPagehit`
- (4) `uint32_t semc_queueb_weight_struct_t::slaveHitNoswitch`
- (5) `uint32_t semc_queueb_weight_struct_t::bankRotation`

**40.4.10 union semc\_queueb\_weight\_t****Data Fields**

- [semc\\_queueb\\_weight\\_struct\\_t queuebConfig](#)  
*Structure configuration for queueB.*
- `uint32_t queuebValue`  
*Configuration value for queueB which could directly write to the reg.*

**Field Documentation**

- (1) `semc_queueb_weight_struct_t semc_queueb_weight_t::queuebConfig`
- (2) `uint32_t semc_queueb_weight_t::queuebValue`

**40.4.11 struct semc\_axi\_queueweight\_t****Data Fields**

- `bool queueaEnable`  
*Enable queue a.*
- [semc\\_queuea\\_weight\\_t queueaWeight](#)  
*Weight settings for queue a.*
- `bool queuebEnable`  
*Enable queue b.*
- [semc\\_queueb\\_weight\\_t queuebWeight](#)  
*Weight settings for queue b.*

### Field Documentation

- (1) `bool semc_axi_queueweight_t::queueaEnable`
- (2) `semc_queuea_weight_t semc_axi_queueweight_t::queueaWeight`
- (3) `bool semc_axi_queueweight_t::queuebEnable`
- (4) `semc_queueb_weight_t semc_axi_queueweight_t::queuebWeight`

### 40.4.12 struct semc\_config\_t

`busTimeoutCycles`: when `busTimeoutCycles` is zero, the bus timeout cycle is  $255 \times 1024$ . otherwise the bus timeout cycles is  $\text{busTimeoutCycles} \times 1024$ . `cmdTimeoutCycles`: is used for command execution timeout cycles. it's similar to the `busTimeoutCycles`.

### Data Fields

- [semc\\_dqs\\_mode\\_t dqsMode](#)  
*Dummy read strobe mode: use enum in "semc\_dqs\_mode\_t".*
- `uint8_t cmdTimeoutCycles`  
*Command execution timeout cycles.*
- `uint8_t busTimeoutCycles`  
*Bus timeout cycles.*
- [semc\\_axi\\_queueweight\\_t queueWeight](#)  
*AXI queue weight.*

### Field Documentation

- (1) `semc_dqs_mode_t semc_config_t::dqsMode`
- (2) `uint8_t semc_config_t::cmdTimeoutCycles`
- (3) `uint8_t semc_config_t::busTimeoutCycles`
- (4) `semc_axi_queueweight_t semc_config_t::queueWeight`

## 40.5 Macro Definition Documentation

40.5.1 `#define FSL_SEMC_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))`

## 40.6 Enumeration Type Documentation

### 40.6.1 anonymous enum

Enumerator

*kStatus\_SEMC\_InvalidDeviceType* Invalid device type.

*kStatus\_SEMC\_IpCommandExecutionError* IP command execution error.

*kStatus\_SEMC\_AxiCommandExecutionError* AXI command execution error.  
*kStatus\_SEMC\_InvalidMemorySize* Invalid memory size.  
*kStatus\_SEMC\_InvalidIpcmdDataSize* Invalid IP command data size.  
*kStatus\_SEMC\_InvalidAddressPortWidth* Invalid address port width.  
*kStatus\_SEMC\_InvalidDataPortWidth* Invalid data port width.  
*kStatus\_SEMC\_InvalidSwPinmuxSelection* Invalid SW pinmux selection.  
*kStatus\_SEMC\_InvalidBurstLength* Invalid burst length.  
*kStatus\_SEMC\_InvalidColumnAddressBitWidth* Invalid column address bit width.  
*kStatus\_SEMC\_InvalidBaseAddress* Invalid base address.  
*kStatus\_SEMC\_InvalidTimerSetting* Invalid timer setting.

#### 40.6.2 enum semc\_mem\_type\_t

Enumerator

*kSEMC\_MemType\_SDRAM* SDRAM.  
*kSEMC\_MemType\_SRAM* SRAM.  
*kSEMC\_MemType\_NOR* NOR.  
*kSEMC\_MemType\_NAND* NAND.  
*kSEMC\_MemType\_8080* 1.

#### 40.6.3 enum semc\_waitready\_polarity\_t

Enumerator

*kSEMC\_LowActive* Low active.  
*kSEMC\_HighActive* High active.

#### 40.6.4 enum semc\_sdram\_cs\_t

Enumerator

*kSEMC\_SDRAM\_CS0* SEMC SDRAM CS0.  
*kSEMC\_SDRAM\_CS1* SEMC SDRAM CS1.  
*kSEMC\_SDRAM\_CS2* SEMC SDRAM CS2.  
*kSEMC\_SDRAM\_CS3* SEMC SDRAM CS3.

#### 40.6.5 enum semc\_sram\_cs\_t

Enumerator

*kSEMC\_SRAM\_CS0* SEMC SRAM CS0.

#### 40.6.6 enum semc\_nand\_access\_type\_t

Enumerator

- kSEMC\_NAND\_ACCESS\_BY\_AXI* Access to NAND flash by AXI bus.
- kSEMC\_NAND\_ACCESS\_BY\_IPCMD* Access to NAND flash by IP bus.

#### 40.6.7 enum semc\_interrupt\_enable\_t

Enumerator

- kSEMC\_IPCmdDoneInterrupt* Ip command done interrupt.
- kSEMC\_IPCmdErrInterrupt* Ip command error interrupt.
- kSEMC\_AXICmdErrInterrupt* AXI command error interrupt.
- kSEMC\_AXIBusErrInterrupt* AXI bus error interrupt.

#### 40.6.8 enum semc\_ipcmd\_datasize\_t

Enumerator

- kSEMC\_IPcmdDataSize\_1bytes* The IP command data size 1 byte.
- kSEMC\_IPcmdDataSize\_2bytes* The IP command data size 2 byte.
- kSEMC\_IPcmdDataSize\_3bytes* The IP command data size 3 byte.
- kSEMC\_IPcmdDataSize\_4bytes* The IP command data size 4 byte.

#### 40.6.9 enum semc\_refresh\_time\_t

Enumerator

- kSEMC\_RefreshThreeClocks* The refresh timing with three bus clocks.
- kSEMC\_RefreshSixClocks* The refresh timing with six bus clocks.
- kSEMC\_RefreshNineClocks* The refresh timing with nine bus clocks.

#### 40.6.10 enum semc\_caslatency\_t

Enumerator

- kSEMC\_LatencyOne* Latency 1.
- kSEMC\_LatencyTwo* Latency 2.
- kSEMC\_LatencyThree* Latency 3.

**40.6.11 enum semc\_sdram\_column\_bit\_num\_t**

Enumerator

*kSEMC\_SdramColumn\_12bit* 12 bit.  
*kSEMC\_SdramColumn\_11bit* 11 bit.  
*kSEMC\_SdramColumn\_10bit* 10 bit.  
*kSEMC\_SdramColumn\_9bit* 9 bit.

**40.6.12 enum sem\_sdram\_burst\_len\_t**

Enumerator

*kSEMC\_Sdram\_BurstLen1* According to ERR050577, Auto-refresh command may possibly fail to be triggered during long time back-to-back write (or read) when SDRAM controller's burst length is greater than 1. Burst length 1  
*kSEMC\_Sdram\_BurstLen2* Burst length 2.  
*kSEMC\_Sdram\_BurstLen4* Burst length 4.  
*kSEMC\_Sdram\_BurstLen8* Burst length 8.

**40.6.13 enum semc\_nand\_column\_bit\_num\_t**

Enumerator

*kSEMC\_NandColum\_16bit* 16 bit.  
*kSEMC\_NandColum\_15bit* 15 bit.  
*kSEMC\_NandColum\_14bit* 14 bit.  
*kSEMC\_NandColum\_13bit* 13 bit.  
*kSEMC\_NandColum\_12bit* 12 bit.  
*kSEMC\_NandColum\_11bit* 11 bit.  
*kSEMC\_NandColum\_10bit* 10 bit.  
*kSEMC\_NandColum\_9bit* 9 bit.

**40.6.14 enum sem\_nand\_burst\_len\_t**

Enumerator

*kSEMC\_Nand\_BurstLen1* Burst length 1.  
*kSEMC\_Nand\_BurstLen2* Burst length 2.  
*kSEMC\_Nand\_BurstLen4* Burst length 4.  
*kSEMC\_Nand\_BurstLen8* Burst length 8.  
*kSEMC\_Nand\_BurstLen16* Burst length 16.

*kSEMC\_Nand\_BurstLen32* Burst length 32.

*kSEMC\_Nand\_BurstLen64* Burst length 64.

#### 40.6.15 enum semc\_norsram\_column\_bit\_num\_t

Enumerator

*kSEMC\_NorColum\_12bit* 12 bit.

*kSEMC\_NorColum\_11bit* 11 bit.

*kSEMC\_NorColum\_10bit* 10 bit.

*kSEMC\_NorColum\_9bit* 9 bit.

*kSEMC\_NorColum\_8bit* 8 bit.

*kSEMC\_NorColum\_7bit* 7 bit.

*kSEMC\_NorColum\_6bit* 6 bit.

*kSEMC\_NorColum\_5bit* 5 bit.

*kSEMC\_NorColum\_4bit* 4 bit.

*kSEMC\_NorColum\_3bit* 3 bit.

*kSEMC\_NorColum\_2bit* 2 bit.

#### 40.6.16 enum sem\_norsram\_burst\_len\_t

Enumerator

*kSEMC\_Nor\_BurstLen1* Burst length 1.

*kSEMC\_Nor\_BurstLen2* Burst length 2.

*kSEMC\_Nor\_BurstLen4* Burst length 4.

*kSEMC\_Nor\_BurstLen8* Burst length 8.

*kSEMC\_Nor\_BurstLen16* Burst length 16.

*kSEMC\_Nor\_BurstLen32* Burst length 32.

*kSEMC\_Nor\_BurstLen64* Burst length 64.

#### 40.6.17 enum semc\_dbi\_column\_bit\_num\_t

Enumerator

*kSEMC\_Dbi\_Colum\_12bit* 12 bit.

*kSEMC\_Dbi\_Colum\_11bit* 11 bit.

*kSEMC\_Dbi\_Colum\_10bit* 10 bit.

*kSEMC\_Dbi\_Colum\_9bit* 9 bit.

*kSEMC\_Dbi\_Colum\_8bit* 8 bit.

*kSEMC\_Dbi\_Colum\_7bit* 7 bit.

*kSEMC\_Dbi\_Colum\_6bit* 6 bit.  
*kSEMC\_Dbi\_Colum\_5bit* 5 bit.  
*kSEMC\_Dbi\_Colum\_4bit* 4 bit.  
*kSEMC\_Dbi\_Colum\_3bit* 3 bit.  
*kSEMC\_Dbi\_Colum\_2bit* 2 bit.

#### 40.6.18 enum sem\_dbi\_burst\_len\_t

Enumerator

*kSEMC\_Dbi\_BurstLen1* Burst length 1.  
*kSEMC\_Dbi\_BurstLen2* Burst length 2.  
*kSEMC\_Dbi\_Dbi\_BurstLen4* Burst length 4.  
*kSEMC\_Dbi\_BurstLen8* Burst length 8.  
*kSEMC\_Dbi\_BurstLen16* Burst length 16.  
*kSEMC\_Dbi\_BurstLen32* Burst length 32.  
*kSEMC\_Dbi\_BurstLen64* Burst length 64.

#### 40.6.19 enum semc\_iomux\_pin

Enumerator

*kSEMC\_MUXA8* MUX A8 pin.  
*kSEMC\_MUXCSX0* MUX CSX0 pin.  
*kSEMC\_MUXCSX1* MUX CSX1 Pin.  
*kSEMC\_MUXCSX2* MUX CSX2 Pin.  
*kSEMC\_MUXCSX3* MUX CSX3 Pin.  
*kSEMC\_MUXRDY* MUX RDY pin.

#### 40.6.20 enum semc\_iomux\_nora27\_pin

Enumerator

*kSEMC\_MORA27\_NONE* No NOR/SRAM A27 pin.  
*kSEMC\_NORA27\_MUXCSX3* MUX CSX3 Pin.  
*kSEMC\_NORA27\_MUXRDY* MUX RDY pin.



#### 40.6.21 enum smec\_port\_size\_t

Enumerator

- kSEMC\_PortSize8Bit* 8-Bit port size.
- kSEMC\_PortSize16Bit* 16-Bit port size.

#### 40.6.22 enum semc\_addr\_mode\_t

Enumerator

- kSEMC\_AddrDataMux* SEMC address/data mux mode.
- kSEMC\_AdvAddrdataMux* Advanced address/data mux mode.
- kSEMC\_AddrDataNonMux* Address/data non-mux mode.

#### 40.6.23 enum semc\_dqs\_mode\_t

Enumerator

- kSEMC\_Loopbackinternal* Dummy read strobe loopbacked internally.
- kSEMC\_Loopbackdqspad* Dummy read strobe loopbacked from DQS pad.

#### 40.6.24 enum semc\_adv\_polarity\_t

Enumerator

- kSEMC\_AdvActiveLow* Adv active low.
- kSEMC\_AdvActiveHigh* Adv active high.

#### 40.6.25 enum semc\_sync\_mode\_t

Enumerator

- kSEMC\_AsyncMode* Async mode.
- kSEMC\_SyncMode* Sync mode.

**40.6.26 enum semc\_adv\_level\_control\_t**

Enumerator

*kSEMC\_AdvHigh* Adv is high during address hold state.*kSEMC\_AdvLow* Adv is low during address hold state.**40.6.27 enum semc\_rdy\_polarity\_t**

Enumerator

*kSEMC\_RdyActiveLow* Adv active low.*kSEMC\_RdyActivehigh* Adv active low.**40.6.28 enum semc\_ipcmd\_nand\_addrmode\_t**

Enumerator

*kSEMC\_NANDAM\_ColumnRow* Address mode: column and row address(5Byte-CA0/CA1/RA0/RA1/RA2).*kSEMC\_NANDAM\_ColumnCA0* Address mode: column address only(1 Byte-CA0).*kSEMC\_NANDAM\_ColumnCA0CA1* Address mode: column address only(2 Byte-CA0/CA1).*kSEMC\_NANDAM\_RawRA0* Address mode: row address only(1 Byte-RA0).*kSEMC\_NANDAM\_RawRA0RA1* Address mode: row address only(2 Byte-RA0/RA1).*kSEMC\_NANDAM\_RawRA0RA1RA2* Address mode: row address only(3 Byte-RA0).**40.6.29 enum semc\_ipcmd\_nand\_cmdmode\_t**

Enumerator

*kSEMC\_NANDCM\_Command* command.*kSEMC\_NANDCM\_CommandHold* Command hold.*kSEMC\_NANDCM\_CommandAddress* Command address.*kSEMC\_NANDCM\_CommandAddressHold* Command address hold.*kSEMC\_NANDCM\_CommandAddressRead* Command address read.*kSEMC\_NANDCM\_CommandAddressWrite* Command address write.*kSEMC\_NANDCM\_CommandRead* Command read.*kSEMC\_NANDCM\_CommandWrite* Command write.*kSEMC\_NANDCM\_Read* Read.*kSEMC\_NANDCM\_Write* Write.

**40.6.30 enum semc\_nand\_address\_option\_t**

Enumerator

*kSEMC\_NandAddrOption\_5byte\_CA2RA3* CA0+CA1+RA0+RA1+RA2.  
*kSEMC\_NandAddrOption\_4byte\_CA2RA2* CA0+CA1+RA0+RA1.  
*kSEMC\_NandAddrOption\_3byte\_CA2RA1* CA0+CA1+RA0.  
*kSEMC\_NandAddrOption\_4byte\_CA1RA3* CA0+RA0+RA1+RA2.  
*kSEMC\_NandAddrOption\_3byte\_CA1RA2* CA0+RA0+RA1.  
*kSEMC\_NandAddrOption\_2byte\_CA1RA1* CA0+RA0.

**40.6.31 enum semc\_ipcmd\_nor\_dbi\_t**

Enumerator

*kSEMC\_NORDBICM\_Read* NOR read.  
*kSEMC\_NORDBICM\_Write* NOR write.

**40.6.32 enum semc\_ipcmd\_sram\_t**

Enumerator

*kSEMC\_SRAMCM\_ArrayRead* SRAM memory array read.  
*kSEMC\_SRAMCM\_ArrayWrite* SRAM memory array write.  
*kSEMC\_SRAMCM\_RegRead* SRAM memory register read.  
*kSEMC\_SRAMCM\_RegWrite* SRAM memory register write.

**40.6.33 enum semc\_ipcmd\_sdram\_t**

Enumerator

*kSEMC\_SDRAMCM\_Read* SDRAM memory read.  
*kSEMC\_SDRAMCM\_Write* SDRAM memory write.  
*kSEMC\_SDRAMCM\_Modeset* SDRAM MODE SET.  
*kSEMC\_SDRAMCM\_Active* SDRAM active.  
*kSEMC\_SDRAMCM\_AutoRefresh* SDRAM auto-refresh.  
*kSEMC\_SDRAMCM\_SelfRefresh* SDRAM self-refresh.  
*kSEMC\_SDRAMCM\_Precharge* SDRAM precharge.  
*kSEMC\_SDRAMCM\_Prechargeall* SDRAM precharge all.

## 40.7 Function Documentation

### 40.7.1 void SEMC\_GetDefaultConfig ( semc\_config\_t \* config )

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

```
semc_config_t config;
SEMC_GetDefaultConfig(&config);
```

Parameters

|               |                                           |
|---------------|-------------------------------------------|
| <i>config</i> | The SEMC configuration structure pointer. |
|---------------|-------------------------------------------|

### 40.7.2 void SEMC\_Init ( SEMC\_Type \* base, semc\_config\_t \* configure )

This function ungates the SEMC clock and initializes SEMC. This function must be called before calling any other SEMC driver functions.

Parameters

|                  |                                           |
|------------------|-------------------------------------------|
| <i>base</i>      | SEMC peripheral base address.             |
| <i>configure</i> | The SEMC configuration structure pointer. |

### 40.7.3 void SEMC\_Deinit ( SEMC\_Type \* base )

This function gates the SEMC clock. As a result, the SEMC module doesn't work after calling this function, for some IDE, calling this API may cause the next downloading operation failed. so, please call this API cautiously. Additional, users can using "#define FSL\_SDK\_DISABLE\_DRIVER\_CLOCK\_CONTROL (1)" to disable the clock control operation in drivers.

Parameters

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

### 40.7.4 status\_t SEMC\_ConfigureSDRAM ( SEMC\_Type \* base, semc\_sdram\_cs\_t cs, semc\_sdram\_config\_t \* config, uint32\_t clkSrc\_Hz )

## Parameters

|                  |                               |
|------------------|-------------------------------|
| <i>base</i>      | SEMC peripheral base address. |
| <i>cs</i>        | The chip selection.           |
| <i>config</i>    | The sdram configuration.      |
| <i>clkSrc_Hz</i> | The SEMC clock frequency.     |

#### 40.7.5 **status\_t SEMC\_ConfigureNAND ( SEMC\_Type \* *base*, semc\_nand\_config\_t \* *config*, uint32\_t *clkSrc\_Hz* )**

## Parameters

|                  |                               |
|------------------|-------------------------------|
| <i>base</i>      | SEMC peripheral base address. |
| <i>config</i>    | The nand configuration.       |
| <i>clkSrc_Hz</i> | The SEMC clock frequency.     |

#### 40.7.6 **status\_t SEMC\_ConfigureNOR ( SEMC\_Type \* *base*, semc\_nor\_config\_t \* *config*, uint32\_t *clkSrc\_Hz* )**

## Parameters

|                  |                               |
|------------------|-------------------------------|
| <i>base</i>      | SEMC peripheral base address. |
| <i>config</i>    | The nor configuration.        |
| <i>clkSrc_Hz</i> | The SEMC clock frequency.     |

#### 40.7.7 **status\_t SEMC\_ConfigureSRAMWithChipSelection ( SEMC\_Type \* *base*, semc\_sram\_cs\_t *cs*, semc\_sram\_config\_t \* *config*, uint32\_t *clkSrc\_Hz* )**

## Parameters

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

|                  |                           |
|------------------|---------------------------|
| <i>cs</i>        | The chip selection.       |
| <i>config</i>    | The sram configuration.   |
| <i>clkSrc_Hz</i> | The SEMC clock frequency. |

#### 40.7.8 `status_t SEMC_ConfigureSRAM ( SEMC_Type * base, semc_sram_config_t * config, uint32_t clkSrc_Hz )`

**Deprecated** Do not use this function. It has been superseded by [SEMC\\_ConfigureSRAMWithChip-Selection](#).

Parameters

|                  |                               |
|------------------|-------------------------------|
| <i>base</i>      | SEMC peripheral base address. |
| <i>config</i>    | The sram configuration.       |
| <i>clkSrc_Hz</i> | The SEMC clock frequency.     |

#### 40.7.9 `status_t SEMC_ConfigureDBI ( SEMC_Type * base, semc_dbi_config_t * config, uint32_t clkSrc_Hz )`

Parameters

|                  |                               |
|------------------|-------------------------------|
| <i>base</i>      | SEMC peripheral base address. |
| <i>config</i>    | The dbi configuration.        |
| <i>clkSrc_Hz</i> | The SEMC clock frequency.     |

#### 40.7.10 `static void SEMC_EnableInterrupts ( SEMC_Type * base, uint32_t mask )` `[inline], [static]`

This function enables the SEMC interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [semc\\_interrupt\\_enable\\_t](#). For example, to enable the IP command done and error interrupt, do the following.

```
* SEMC_EnableInterrupts (ENET, kSEMC_IPCmdDoneInterrupt |
* kSEMC_IPCmdErrInterrupt);
```

## Parameters

|             |                                                                                                              |
|-------------|--------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SEMC peripheral base address.                                                                                |
| <i>mask</i> | SEMC interrupts to enable. This is a logical OR of the enumeration :: <code>semc_interrupt_enable_t</code> . |

#### 40.7.11 `static void SEMC_DisableInterrupts ( SEMC_Type * base, uint32_t mask ) [inline], [static]`

This function disables the SEMC interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [semc\\_interrupt\\_enable\\_t](#). For example, to disable the IP command done and error interrupt, do the following.

```
* SEMC_DisableInterrupts (ENET,
* kSEMC_IPCmdDoneInterrupt | kSEMC_IPCmdErrInterrupt);
*
```

## Parameters

|             |                                                                                                               |
|-------------|---------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SEMC peripheral base address.                                                                                 |
| <i>mask</i> | SEMC interrupts to disable. This is a logical OR of the enumeration :: <code>semc_interrupt_enable_t</code> . |

#### 40.7.12 `static bool SEMC_GetStatusFlag ( SEMC_Type * base ) [inline], [static]`

This function gets the SEMC interrupts event status. User can use the a logical OR of enumeration member as a mask. See [semc\\_interrupt\\_enable\\_t](#).

## Parameters

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

## Returns

status flag, use status flag in `semc_interrupt_enable_t` to get the related status.

#### 40.7.13 `static void SEMC_ClearStatusFlags ( SEMC_Type * base, uint32_t mask ) [inline], [static]`

The following status register flags can be cleared SEMC interrupt status.

Parameters

|             |                                                                                                    |
|-------------|----------------------------------------------------------------------------------------------------|
| <i>base</i> | SEMC base pointer                                                                                  |
| <i>mask</i> | The status flag mask, a logical OR of enumeration member <a href="#">semc_interrupt_enable_t</a> . |

#### 40.7.14 **static bool SEMC\_IsInIdle ( SEMC\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

True SEMC is in idle, false is not in idle.

#### 40.7.15 **status\_t SEMC\_SendIPCommand ( SEMC\_Type \* *base*, semc\_mem\_type\_t *memType*, uint32\_t *address*, uint32\_t *command*, uint32\_t \* *write*, uint32\_t \* *read* )**

Parameters

|                |                                                                                                                                                                                                                                                                                |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | SEMC peripheral base address.                                                                                                                                                                                                                                                  |
| <i>memType</i> | SEMC memory type. refer to "semc_mem_type_t"                                                                                                                                                                                                                                   |
| <i>address</i> | SEMC device address.                                                                                                                                                                                                                                                           |
| <i>command</i> | SEMC IP command. For NAND device, we should use the SEMC_BuildNandIPCommand to get the right nand command. For NOR/DBI device, take refer to "semc_ipcmd_nor_dbi_t". For SRAM device, take refer to "semc_ipcmd_sram_t". For SDRAM device, take refer to "semc_ipcmd_sdram_t". |
| <i>write</i>   | Data for write access.                                                                                                                                                                                                                                                         |
| <i>read</i>    | Data pointer for read data out.                                                                                                                                                                                                                                                |

#### 40.7.16 **static uint16\_t SEMC\_BuildNandIPCommand ( uint8\_t *userCommand*, semc\_ipcmd\_nand\_addrmode\_t *addrMode*, semc\_ipcmd\_nand\_cmdmode\_t *cmdMode* ) [inline], [static]**

This function build SEMC NAND IP command. The command is build of user command code, SEMC address mode and SEMC command mode.



## Parameters

|                    |                                                           |
|--------------------|-----------------------------------------------------------|
| <i>userCommand</i> | NAND device normal command.                               |
| <i>addrMode</i>    | NAND address mode. Refer to "semc_ipcmd_nand_addrmode_t". |
| <i>cmdMode</i>     | NAND command mode. Refer to "semc_ipcmd_nand_cmdmode_t".  |

#### 40.7.17 static bool SEMC\_IsNandReady ( SEMC\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

True NAND is ready, false NAND is not ready.

#### 40.7.18 status\_t SEMC\_IPCommandNandWrite ( SEMC\_Type \* *base*, uint32\_t *address*, uint8\_t \* *data*, uint32\_t *size\_bytes* )

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>base</i>       | SEMC peripheral base address. |
| <i>address</i>    | SEMC NAND device address.     |
| <i>data</i>       | Data for write access.        |
| <i>size_bytes</i> | Data length.                  |

#### 40.7.19 status\_t SEMC\_IPCommandNandRead ( SEMC\_Type \* *base*, uint32\_t *address*, uint8\_t \* *data*, uint32\_t *size\_bytes* )

## Parameters

|                   |                                 |
|-------------------|---------------------------------|
| <i>base</i>       | SEMC peripheral base address.   |
| <i>address</i>    | SEMC NAND device address.       |
| <i>data</i>       | Data pointer for data read out. |
| <i>size_bytes</i> | Data length.                    |

#### 40.7.20 **status\_t SEMC\_IPCommandNorWrite ( SEMC\_Type \* *base*, uint32\_t *address*, uint8\_t \* *data*, uint32\_t *size\_bytes* )**

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>base</i>       | SEMC peripheral base address. |
| <i>address</i>    | SEMC NOR device address.      |
| <i>data</i>       | Data for write access.        |
| <i>size_bytes</i> | Data length.                  |

#### 40.7.21 **status\_t SEMC\_IPCommandNorRead ( SEMC\_Type \* *base*, uint32\_t *address*, uint8\_t \* *data*, uint32\_t *size\_bytes* )**

Parameters

|                   |                                 |
|-------------------|---------------------------------|
| <i>base</i>       | SEMC peripheral base address.   |
| <i>address</i>    | SEMC NOR device address.        |
| <i>data</i>       | Data pointer for data read out. |
| <i>size_bytes</i> | Data length.                    |

# Chapter 41

## SNVS: Secure Non-Volatile Storage

### 41.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Secure Non-Volatile Storage (SNVS) module.

The SNVS module is designed to safely hold security-related data such as cryptographic key, time counter, monotonic counter, and general purpose security information. The SNVS includes a low power section, namely SNVS\_LP, that is battery backed by the SVNS (or VBAT) power domain. This enables it to keep this data valid and continue to increment the time counter when the power goes down in the rest of the SoC. The always-powered-up part of the module is isolated from the rest of the logic to ensure that it does not get corrupted when the SoC is powered down. The SNVS is designed to comply with Digital Rights Management (DRM) and other security application rules and requirements. This trusted hardware provides features that allow the system software designer to ensure that the data kept by the device is certifiable. Specially, it incorporates a security monitor that checks for various security conditions. If a security violation is indicated then it invalidates access to its sensitive data, and the secret data, for instance, Zeroizable Secret Key, is zeroized. the SNVS can be also configured to bypass its security features and protection mechanism. In this case it can be used by systems that do not require security.

### Modules

- [Secure Non-Volatile Storage High-Power](#)
- [Secure Non-Volatile Storage Low-Power](#)

## 41.2 Secure Non-Volatile Storage High-Power

### 41.2.1 Overview

The MCUXpresso SDK provides a Peripheral driver for the Secure Non-Volatile Storage High-Power(S-NVS-HP) module.

The SNVS\_HP is in the chip's power-supply domain and thus receives the power along with the rest of the chip. The SNVS\_HP provides an interface between the SNVS\_LP and the rest of the system; there is no way to access the SNVS\_LP registers except through the SNVS\_HP. For access to the SNVS\_LP registers, the SNVS\_HP must be powered up. It uses a register access permission policy to determine whether the access to the particular registers is permitted.

### Data Structures

- struct `snvs_hp_rtc_datetime_t`  
*Structure is used to hold the date and time. [More...](#)*
- struct `snvs_hp_rtc_config_t`  
*SNVS config structure. [More...](#)*

### Macros

- #define `SNVS_MAKE_HP_SV_FLAG(x)` ( $1U \ll (SNVS_HPSVSR_SV0\_SHIFT + (x))$ )  
*Macro to make security violation flag.*

### Enumerations

- enum `snvs_hp_interrupts_t` {  
`kSNVS_RTC_AlarmInterrupt = SNVS_HPCR_HPTA_EN_MASK,`  
`kSNVS_RTC_PeriodicInterrupt = SNVS_HPCR_PI_EN_MASK }`  
*List of SNVS interrupts.*
- enum `snvs_hp_status_flags_t` {  
`kSNVS_RTC_AlarmInterruptFlag = SNVS_HPSR_HPTA_MASK,`  
`kSNVS_RTC_PeriodicInterruptFlag = SNVS_HPSR_PI_MASK,`  
`kSNVS_ZMK_ZeroFlag = (int)SNVS_HPSR_ZMK_ZERO_MASK,`  
`kSNVS_OTPMK_ZeroFlag = SNVS_HPSR_OTPMK_ZERO_MASK }`  
*List of SNVS flags.*
- enum `snvs_hp_sv_status_flags_t` {

```

kSNVS_LP_ViolationFlag = SNVS_HPSVSR_SW_LPSV_MASK,
kSNVS_ZMK_EccFailFlag = SNVS_HPSVSR_ZMK_ECC_FAIL_MASK,
kSNVS_LP_SoftwareViolationFlag = SNVS_HPSVSR_SW_LPSV_MASK,
kSNVS_FatalSoftwareViolationFlag = SNVS_HPSVSR_SW_FSV_MASK,
kSNVS_SoftwareViolationFlag = SNVS_HPSVSR_SW_SV_MASK,
kSNVS_Violation0Flag = SNVS_HPSVSR_SV0_MASK,
kSNVS_Violation1Flag = SNVS_HPSVSR_SV1_MASK,
kSNVS_Violation2Flag = SNVS_HPSVSR_SV2_MASK,
kSNVS_Violation4Flag = SNVS_HPSVSR_SV4_MASK,
kSNVS_Violation5Flag = SNVS_HPSVSR_SV5_MASK }

```

*List of SNVS security violation flags.*

- enum `snvs_hp_ssm_state_t` {
 

```

kSNVS_SSMInit = 0x00,
kSNVS_SSMHardFail = 0x01,
kSNVS_SSMSoftFail = 0x03,
kSNVS_SSMInitInter = 0x08,
kSNVS_SSMCheck = 0x09,
kSNVS_SSMNonSecure = 0x0B,
kSNVS_SSMTrusted = 0x0D,
kSNVS_SSMSecure = 0x0F }

```

*List of SNVS Security State Machine State.*

## Functions

- static void `SNVS_HP_EnableMasterKeySelection` (SNVS\_Type \*base, bool enable)
 

*Enable or disable master key selection.*
- static void `SNVS_HP_ProgramZeroizableMasterKey` (SNVS\_Type \*base)
 

*Trigger to program Zeroizable Master Key.*
- static void `SNVS_HP_ChangeSSMState` (SNVS\_Type \*base)
 

*Trigger SSM State Transition.*
- static void `SNVS_HP_SetSoftwareFatalSecurityViolation` (SNVS\_Type \*base)
 

*Trigger Software Fatal Security Violation.*
- static void `SNVS_HP_SetSoftwareSecurityViolation` (SNVS\_Type \*base)
 

*Trigger Software Security Violation.*
- static `snvs_hp_ssm_state_t` `SNVS_HP_GetSSMState` (SNVS\_Type \*base)
 

*Get current SSM State.*
- static void `SNVS_HP_ResetLP` (SNVS\_Type \*base)
 

*Reset the SNVS LP section.*
- static `uint32_t` `SNVS_HP_GetStatusFlags` (SNVS\_Type \*base)
 

*Get the SNVS HP status flags.*
- static void `SNVS_HP_ClearStatusFlags` (SNVS\_Type \*base, `uint32_t` mask)
 

*Clear the SNVS HP status flags.*
- static `uint32_t` `SNVS_HP_GetSecurityViolationStatusFlags` (SNVS\_Type \*base)
 

*Get the SNVS HP security violation status flags.*
- static void `SNVS_HP_ClearSecurityViolationStatusFlags` (SNVS\_Type \*base, `uint32_t` mask)
 

*Clear the SNVS HP security violation status flags.*

## Driver version

- #define `FSL_SNVS_HP_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 2)`)  
*Version 2.3.2.*

## Initialization and deinitialization

- void `SNVS_HP_Init` (`SNVS_Type *base`)  
*Initialize the SNVS.*
- void `SNVS_HP_Deinit` (`SNVS_Type *base`)  
*Deinitialize the SNVS.*
- void `SNVS_HP_RTC_Init` (`SNVS_Type *base`, const `snvs_hp_rtc_config_t *config`)  
*Ungates the SNVS clock and configures the peripheral for basic operation.*
- void `SNVS_HP_RTC_Deinit` (`SNVS_Type *base`)  
*Stops the RTC and SRTC timers.*
- void `SNVS_HP_RTC_GetDefaultConfig` (`snvs_hp_rtc_config_t *config`)  
*Fills in the SNVS config struct with the default settings.*

## Non secure RTC current Time & Alarm

- `status_t SNVS_HP_RTC_SetDatetime` (`SNVS_Type *base`, const `snvs_hp_rtc_datetime_t *datetime`)  
*Sets the SNVS RTC date and time according to the given time structure.*
- void `SNVS_HP_RTC_GetDatetime` (`SNVS_Type *base`, `snvs_hp_rtc_datetime_t *datetime`)  
*Gets the SNVS RTC time and stores it in the given time structure.*
- `status_t SNVS_HP_RTC_SetAlarm` (`SNVS_Type *base`, const `snvs_hp_rtc_datetime_t *alarmTime`)  
*Sets the SNVS RTC alarm time.*
- void `SNVS_HP_RTC_GetAlarm` (`SNVS_Type *base`, `snvs_hp_rtc_datetime_t *datetime`)  
*Returns the SNVS RTC alarm time.*

## Interrupt Interface

- static void `SNVS_HP_RTC_EnableInterrupts` (`SNVS_Type *base`, `uint32_t mask`)  
*Enables the selected SNVS interrupts.*
- static void `SNVS_HP_RTC_DisableInterrupts` (`SNVS_Type *base`, `uint32_t mask`)  
*Disables the selected SNVS interrupts.*
- `uint32_t SNVS_HP_RTC_GetEnabledInterrupts` (`SNVS_Type *base`)  
*Gets the enabled SNVS interrupts.*

## Status Interface

- `uint32_t SNVS_HP_RTC_GetStatusFlags` (`SNVS_Type *base`)  
*Gets the SNVS status flags.*
- static void `SNVS_HP_RTC_ClearStatusFlags` (`SNVS_Type *base`, `uint32_t mask`)

*Clears the SNVS status flags.*

## Timer Start and Stop

- static void [SNVS\\_HP\\_RTC\\_StartTimer](#) (SNVS\_Type \*base)  
*Starts the SNVS RTC time counter.*
- static void [SNVS\\_HP\\_RTC\\_StopTimer](#) (SNVS\_Type \*base)  
*Stops the SNVS RTC time counter.*

## High Assurance Counter (HAC)

- static void [SNVS\\_HP\\_EnableHighAssuranceCounter](#) (SNVS\_Type \*base, bool enable)  
*Enable or disable the High Assurance Counter (HAC)*
- static void [SNVS\\_HP\\_StartHighAssuranceCounter](#) (SNVS\_Type \*base, bool start)  
*Start or stop the High Assurance Counter (HAC)*
- static void [SNVS\\_HP\\_SetHighAssuranceCounterInitialValue](#) (SNVS\_Type \*base, uint32\_t value)  
*Set the High Assurance Counter (HAC) initialize value.*
- static void [SNVS\\_HP\\_LoadHighAssuranceCounter](#) (SNVS\_Type \*base)  
*Load the High Assurance Counter (HAC)*
- static uint32\_t [SNVS\\_HP\\_GetHighAssuranceCounter](#) (SNVS\_Type \*base)  
*Get the current High Assurance Counter (HAC) value.*
- static void [SNVS\\_HP\\_ClearHighAssuranceCounter](#) (SNVS\_Type \*base)  
*Clear the High Assurance Counter (HAC)*
- static void [SNVS\\_HP\\_LockHighAssuranceCounter](#) (SNVS\_Type \*base)  
*Lock the High Assurance Counter (HAC)*

## 41.2.2 Data Structure Documentation

### 41.2.2.1 struct snvs\_hp\_rtc\_datetime\_t

#### Data Fields

- uint16\_t [year](#)  
*Range from 1970 to 2099.*
- uint8\_t [month](#)  
*Range from 1 to 12.*
- uint8\_t [day](#)  
*Range from 1 to 31 (depending on month).*
- uint8\_t [hour](#)  
*Range from 0 to 23.*
- uint8\_t [minute](#)  
*Range from 0 to 59.*
- uint8\_t [second](#)  
*Range from 0 to 59.*

**Field Documentation**

- (1) `uint16_t snvs_hp_rtc_datetime_t::year`
- (2) `uint8_t snvs_hp_rtc_datetime_t::month`
- (3) `uint8_t snvs_hp_rtc_datetime_t::day`
- (4) `uint8_t snvs_hp_rtc_datetime_t::hour`
- (5) `uint8_t snvs_hp_rtc_datetime_t::minute`
- (6) `uint8_t snvs_hp_rtc_datetime_t::second`

**41.2.2.2 struct snvs\_hp\_rtc\_config\_t**

This structure holds the configuration settings for the SNVS peripheral. To initialize this structure to reasonable defaults, call the `SNVS_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 `rtcCalEnable`  
*true: RTC calibration mechanism is enabled; false: No calibration is used*
- `uint32_t rtcCalValue`  
*Defines signed calibration value for nonsecure RTC; This is a 5-bit 2's complement value, range from -16 to +15.*
- `uint32_t periodicInterruptFreq`  
*Defines frequency of the periodic interrupt; Range from 0 to 15.*

**41.2.3 Macro Definition Documentation****41.2.3.1 #define SNVS\_MAKE\_HP\_SV\_FLAG( x ) (1U << (SNVS\_HPSVSR\_SV0\_SHIFT + (x)))**

Macro help to make security violation flag `kSNVS_Violation0Flag` to `kSNVS_Violation5Flag`, For example, `SNVS_MAKE_HP_SV_FLAG(0)` is `kSNVS_Violation0Flag`.



## 41.2.4 Enumeration Type Documentation

### 41.2.4.1 enum snvs\_hp\_interrupts\_t

Enumerator

*kSNVS\_RTC\_AlarmInterrupt* RTC time alarm.  
*kSNVS\_RTC\_PeriodicInterrupt* RTC periodic interrupt.

### 41.2.4.2 enum snvs\_hp\_status\_flags\_t

Enumerator

*kSNVS\_RTC\_AlarmInterruptFlag* RTC time alarm flag.  
*kSNVS\_RTC\_PeriodicInterruptFlag* RTC periodic interrupt flag.  
*kSNVS\_ZMK\_ZeroFlag* The ZMK is zero.  
*kSNVS\_OTPMK\_ZeroFlag* The OTPMK is zero.

### 41.2.4.3 enum snvs\_hp\_sv\_status\_flags\_t

Enumerator

*kSNVS\_LP\_ViolationFlag* Low Power section Security Violation.  
*kSNVS\_ZMK\_EccFailFlag* Zeroizable Master Key Error Correcting Code Check Failure.  
*kSNVS\_LP\_SoftwareViolationFlag* LP Software Security Violation.  
*kSNVS\_FatalSoftwareViolationFlag* Software Fatal Security Violation.  
*kSNVS\_SoftwareViolationFlag* Software Security Violation.  
*kSNVS\_Violation0Flag* Security Violation 0.  
*kSNVS\_Violation1Flag* Security Violation 1.  
*kSNVS\_Violation2Flag* Security Violation 2.  
*kSNVS\_Violation4Flag* Security Violation 4.  
*kSNVS\_Violation5Flag* Security Violation 5.

### 41.2.4.4 enum snvs\_hp\_ssm\_state\_t

Enumerator

*kSNVS\_SSMInit* Init.  
*kSNVS\_SSMHardFail* Hard Fail.  
*kSNVS\_SSMSoftFail* Soft Fail.  
*kSNVS\_SSMInitInter* Init Intermediate (transition state between Init and Check)  
*kSNVS\_SSMCheck* Check.  
*kSNVS\_SSMNonSecure* Non-Secure.  
*kSNVS\_SSMTrusted* Trusted.  
*kSNVS\_SSMSecure* Secure.

## 41.2.5 Function Documentation

### 41.2.5.1 void SNVS\_HP\_Init ( SNVS\_Type \* *base* )

Note

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

Parameters

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

### 41.2.5.2 void SNVS\_HP\_Deinit ( SNVS\_Type \* *base* )

Parameters

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

### 41.2.5.3 void SNVS\_HP\_RTC\_Init ( SNVS\_Type \* *base*, const snvs\_hp\_rtc\_config\_t \* *config* )

Note

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

Parameters

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

### 41.2.5.4 void SNVS\_HP\_RTC\_Deinit ( SNVS\_Type \* *base* )

Parameters

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

### 41.2.5.5 void SNVS\_HP\_RTC\_GetDefaultConfig ( snvs\_hp\_rtc\_config\_t \* *config* )

The default values are as follows.

```

* config->rtccalenable = false;
* config->rtccalvalue = 0U;
* config->PIFreq = 0U;
*

```

## Parameters

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

#### 41.2.5.6 **status\_t SNVS\_HP\_RTC\_SetDatetime ( SNVS\_Type \* *base*, const snvs\_hp\_rtc\_datetime\_t \* *datetime* )**

## Parameters

|                 |                                                                      |
|-----------------|----------------------------------------------------------------------|
| <i>base</i>     | SNVS 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 SNVS RTC  
kStatus\_InvalidArgument: Error because the datetime format is incorrect

#### 41.2.5.7 **void SNVS\_HP\_RTC\_GetDatetime ( SNVS\_Type \* *base*, snvs\_hp\_rtc\_datetime\_t \* *datetime* )**

## Parameters

|                 |                                                                      |
|-----------------|----------------------------------------------------------------------|
| <i>base</i>     | SNVS peripheral base address                                         |
| <i>datetime</i> | Pointer to the structure where the date and time details are stored. |

#### 41.2.5.8 **status\_t SNVS\_HP\_RTC\_SetAlarm ( SNVS\_Type \* *base*, const snvs\_hp\_rtc\_datetime\_t \* *alarmTime* )**

The function sets the RTC alarm. It also 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>      | SNVS peripheral base address                             |
| <i>alarmTime</i> | Pointer to the structure where the alarm time is stored. |

## Returns

kStatus\_Success: success in setting the SNVS RTC alarm  
 kStatus\_InvalidArgument: Error because the alarm datetime format is incorrect  
 kStatus\_Fail: Error because the alarm time has already passed

#### 41.2.5.9 void SNVS\_HP\_RTC\_GetAlarm ( SNVS\_Type \* *base*, snvs\_hp\_rtc\_datetime\_t \* *datetime* )

## Parameters

|                 |                                                                            |
|-----------------|----------------------------------------------------------------------------|
| <i>base</i>     | SNVS peripheral base address                                               |
| <i>datetime</i> | Pointer to the structure where the alarm date and time details are stored. |

#### 41.2.5.10 static void SNVS\_HP\_RTC\_EnableInterrupts ( SNVS\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                                                                    |
|-------------|--------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SNVS peripheral base address                                                                                       |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration :: <code>_snvs_hp_interrupts_t</code> |

#### 41.2.5.11 static void SNVS\_HP\_RTC\_DisableInterrupts ( SNVS\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

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

|             |                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------|
| <i>mask</i> | The interrupts to disable. This is a logical OR of members of the enumeration :: <code>_snvs_hp_interrupts_t</code> |
|-------------|---------------------------------------------------------------------------------------------------------------------|

#### 41.2.5.12 `uint32_t SNVS_HP_RTC_GetEnabledInterrupts ( SNVS_Type * base )`

Parameters

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

Returns

The enabled interrupts. This is the logical OR of members of the enumeration :: `_snvs_hp_interrupts_t`

#### 41.2.5.13 `uint32_t SNVS_HP_RTC_GetStatusFlags ( SNVS_Type * base )`

Parameters

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

Returns

The status flags. This is the logical OR of members of the enumeration :: `_snvs_hp_status_flags_t`

#### 41.2.5.14 `static void SNVS_HP_RTC_ClearStatusFlags ( SNVS_Type * base, uint32_t mask ) [inline], [static]`

Parameters

|             |                                                                                                                       |
|-------------|-----------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SNVS peripheral base address                                                                                          |
| <i>mask</i> | The status flags to clear. This is a logical OR of members of the enumeration :: <code>_snvs_hp_status_flags_t</code> |

#### 41.2.5.15 `static void SNVS_HP_RTC_StartTimer ( SNVS_Type * base ) [inline], [static]`

Parameters

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

**41.2.5.16** `static void SNVS_HP_RTC_StopTimer ( SNVS_Type * base ) [inline], [static]`

Parameters

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

**41.2.5.17** `static void SNVS_HP_EnableMasterKeySelection ( SNVS_Type * base, bool enable ) [inline], [static]`

Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | SNVS peripheral base address           |
| <i>enable</i> | Pass true to enable, false to disable. |

**41.2.5.18** `static void SNVS_HP_ProgramZeroizableMasterKey ( SNVS_Type * base ) [inline], [static]`

Parameters

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

**41.2.5.19** `static void SNVS_HP_ChangeSSMState ( SNVS_Type * base ) [inline], [static]`

Trigger state transition of the system security monitor (SSM). It results only the following transitions of the SSM:

- Check State -> Non-Secure (when Non-Secure Boot and not in Fab Configuration)
- Check State -> Trusted (when Secure Boot or in Fab Configuration)
- Trusted State -> Secure
- Secure State -> Trusted
- Soft Fail -> Non-Secure

Parameters

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

**41.2.5.20 static void SNVS\_HP\_SetSoftwareFatalSecurityViolation ( SNVS\_Type \* *base* )**  
**[inline], [static]**

The result SSM state transition is:

- Check State -> Soft Fail
- Non-Secure State -> Soft Fail
- Trusted State -> Soft Fail
- Secure State -> Soft Fail

Parameters

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

**41.2.5.21 static void SNVS\_HP\_SetSoftwareSecurityViolation ( SNVS\_Type \* *base* )**  
**[inline], [static]**

The result SSM state transition is:

- Check -> Non-Secure
- Trusted -> Soft Fail
- Secure -> Soft Fail

Parameters

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

**41.2.5.22 static snvs\_hp\_ssm\_state\_t SNVS\_HP\_GetSSMState ( SNVS\_Type \* *base* )**  
**[inline], [static]**

Parameters

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

Returns

Current SSM state

**41.2.5.23** `static void SNVS_HP_ResetLP ( SNVS_Type * base ) [inline], [static]`

Reset the LP section except SRTC and Time alarm.



Parameters

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

**41.2.5.24** `static void SNVS_HP_EnableHighAssuranceCounter ( SNVS_Type * base, bool enable ) [inline], [static]`

Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | SNVS peripheral base address           |
| <i>enable</i> | Pass true to enable, false to disable. |

**41.2.5.25** `static void SNVS_HP_StartHighAssuranceCounter ( SNVS_Type * base, bool start ) [inline], [static]`

Parameters

|              |                                    |
|--------------|------------------------------------|
| <i>base</i>  | SNVS peripheral base address       |
| <i>start</i> | Pass true to start, false to stop. |

**41.2.5.26** `static void SNVS_HP_SetHighAssuranceCounterInitialValue ( SNVS_Type * base, uint32_t value ) [inline], [static]`

Parameters

|              |                              |
|--------------|------------------------------|
| <i>base</i>  | SNVS peripheral base address |
| <i>value</i> | The initial value to set.    |

**41.2.5.27** `static void SNVS_HP_LoadHighAssuranceCounter ( SNVS_Type * base ) [inline], [static]`

This function loads the HAC initialize value to counter register.

Parameters

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

41.2.5.28 `static uint32_t SNVS_HP_GetHighAssuranceCounter ( SNVS_Type * base )`  
`[inline], [static]`

Parameters

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

Returns

HAC current value.

#### 41.2.5.29 **static void SNVS\_HP\_ClearHighAssuranceCounter ( SNVS\_Type \* *base* ) [inline], [static]**

This function can be called in a functional or soft fail state. When the HAC is enabled:

- If the HAC is cleared in the soft fail state, the SSM transitions to the hard fail state immediately;
- If the HAC is cleared in functional state, the SSM will transition to hard fail immediately after transitioning to soft fail.

Parameters

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

#### 41.2.5.30 **static void SNVS\_HP\_LockHighAssuranceCounter ( SNVS\_Type \* *base* ) [inline], [static]**

Once locked, the HAC initialize value could not be changed, the HAC enable status could not be changed. This could only be unlocked by system reset.

Parameters

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

#### 41.2.5.31 **static uint32\_t SNVS\_HP\_GetStatusFlags ( SNVS\_Type \* *base* ) [inline], [static]**

The flags are returned as the OR'ed value of the enumeration :: `_snvs_hp_status_flags_t`.

Parameters

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

Returns

The OR'ed value of status flags.

**41.2.5.32** `static void SNVS_HP_ClearStatusFlags ( SNVS_Type * base, uint32_t mask )`  
`[inline], [static]`

The flags to clear are passed in as the OR'ed value of the enumeration :: `_snvs_hp_status_flags_t`. Only these flags could be cleared using this API.

- [kSNVS\\_RTC\\_PeriodicInterruptFlag](#)
- [kSNVS\\_RTC\\_AlarmInterruptFlag](#)

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | SNVS peripheral base address       |
| <i>mask</i> | OR'ed value of the flags to clear. |

**41.2.5.33** `static uint32_t SNVS_HP_GetSecurityViolationStatusFlags ( SNVS_Type * base`  
`) [inline], [static]`

The flags are returned as the OR'ed value of the enumeration :: `_snvs_hp_sv_status_flags_t`.

Parameters

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

Returns

The OR'ed value of security violation status flags.

**41.2.5.34** `static void SNVS_HP_ClearSecurityViolationStatusFlags ( SNVS_Type * base,`  
`uint32_t mask ) [inline], [static]`

The flags to clear are passed in as the OR'ed value of the enumeration :: `_snvs_hp_sv_status_flags_t`. Only these flags could be cleared using this API.

- [kSNVS\\_ZMK\\_EccFailFlag](#)
- [kSNVS\\_Violation0Flag](#)
- [kSNVS\\_Violation1Flag](#)
- [kSNVS\\_Violation2Flag](#)
- [kSNVS\\_Violation3Flag](#)
- [kSNVS\\_Violation4Flag](#)
- [kSNVS\\_Violation5Flag](#)

### Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | SNVS peripheral base address       |
| <i>mask</i> | OR'ed value of the flags to clear. |

## 41.3 Secure Non-Volatile Storage Low-Power

### 41.3.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Secure Non-Volatile Storage Low-Power (SNVS-LP) module.

The SNVS\_LP is a data storage subsystem. Its purpose is to store and protect system data, regardless of the main system power state. The SNVS\_LP is in the always-powered-up domain, which is a separate power domain with its own power supply.

### Data Structures

- struct [snvs\\_lp\\_passive\\_tamper\\_t](#)  
*Structure is used to configure SNVS LP passive tamper pins. [More...](#)*
- struct [snvs\\_lp\\_srtc\\_datetime\\_t](#)  
*Structure is used to hold the date and time. [More...](#)*
- struct [snvs\\_lp\\_srtc\\_config\\_t](#)  
*SNVS\_LP config structure. [More...](#)*

### Macros

- #define [SNVS\\_ZMK\\_REG\\_COUNT](#) 8U /\* 8 Zeroizable Master Key registers. \*/  
*Define of SNVS\_LP Zeroizable Master Key registers.*
- #define [SNVS\\_LP\\_MAX\\_TAMPER](#) kSNVS\_ExternalTamper1  
*Define of SNVS\_LP Max possible tamper.*

### Enumerations

- enum [snvs\\_lp\\_srtc\\_interrupts\\_t](#) { kSNVS\_SRTC\_AlarmInterrupt = SNVS\_LPCR\_LPTA\_EN\_MASK }
- enum [snvs\\_lp\\_srtc\\_status\\_flags\\_t](#) { kSNVS\_SRTC\_AlarmInterruptFlag = SNVS\_LPSR\_LPTA\_MASK }
- enum [snvs\\_lp\\_external\\_tamper\\_status\\_t](#)  
*List of SNVS\_LP external tamper status.*
- enum [snvs\\_lp\\_external\\_tamper\\_polarity\\_t](#)  
*SNVS\_LP external tamper polarity.*
- enum [snvs\\_lp\\_zmk\\_program\\_mode\\_t](#) { kSNVS\_ZMKSoftwareProgram, kSNVS\_ZMKHardwareProgram }
- enum [snvs\\_lp\\_master\\_key\\_mode\\_t](#) { kSNVS\_OTPMK = 0, kSNVS\_ZMK = 2, }

```
kSNVS_CMK = 3 }
SNVS_LP Master Key mode.
```

## Functions

- void [SNVS\\_LP\\_SRTC\\_Init](#) (SNVS\_Type \*base, const [snvs\\_lp\\_srtc\\_config\\_t](#) \*config)  
*Ungates the SNVS clock and configures the peripheral for basic operation.*
- void [SNVS\\_LP\\_SRTC\\_Deinit](#) (SNVS\_Type \*base)  
*Stops the SRTC timer.*
- void [SNVS\\_LP\\_SRTC\\_GetDefaultConfig](#) ([snvs\\_lp\\_srtc\\_config\\_t](#) \*config)  
*Fills in the SNVS\_LP config struct with the default settings.*

## Driver version

- #define [FSL\\_SNVS\\_LP\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 4, 5))  
*Version 2.4.5.*

## Initialization and deinitialization

- void [SNVS\\_LP\\_Init](#) (SNVS\_Type \*base)  
*Ungates the SNVS clock and configures the peripheral for basic operation.*
- void [SNVS\\_LP\\_Deinit](#) (SNVS\_Type \*base)  
*Deinit the SNVS LP section.*

## Secure RTC (SRTC) current Time & Alarm

- [status\\_t](#) [SNVS\\_LP\\_SRTC\\_SetDatetime](#) (SNVS\_Type \*base, const [snvs\\_lp\\_srtc\\_datetime\\_t](#) \*datetime)  
*Sets the SNVS SRTC date and time according to the given time structure.*
- void [SNVS\\_LP\\_SRTC\\_GetDatetime](#) (SNVS\_Type \*base, [snvs\\_lp\\_srtc\\_datetime\\_t](#) \*datetime)  
*Gets the SNVS SRTC time and stores it in the given time structure.*
- [status\\_t](#) [SNVS\\_LP\\_SRTC\\_SetAlarm](#) (SNVS\_Type \*base, const [snvs\\_lp\\_srtc\\_datetime\\_t](#) \*alarm-Time)  
*Sets the SNVS SRTC alarm time.*
- void [SNVS\\_LP\\_SRTC\\_GetAlarm](#) (SNVS\_Type \*base, [snvs\\_lp\\_srtc\\_datetime\\_t](#) \*datetime)  
*Returns the SNVS SRTC alarm time.*

## Interrupt Interface

- static void [SNVS\\_LP\\_SRTC\\_EnableInterrupts](#) (SNVS\_Type \*base, uint32\_t mask)  
*Enables the selected SNVS interrupts.*
- static void [SNVS\\_LP\\_SRTC\\_DisableInterrupts](#) (SNVS\_Type \*base, uint32\_t mask)  
*Disables the selected SNVS interrupts.*
- uint32\_t [SNVS\\_LP\\_SRTC\\_GetEnabledInterrupts](#) (SNVS\_Type \*base)

*Gets the enabled SNVS interrupts.*

## Status Interface

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

## Timer Start and Stop

- static void [SNVS\\_LP\\_SRTC\\_StartTimer](#) (SNVS\_Type \*base)  
*Starts the SNVS SRTC time counter.*
- static void [SNVS\\_LP\\_SRTC\\_StopTimer](#) (SNVS\_Type \*base)  
*Stops the SNVS SRTC time counter.*

## External tampering

- void [SNVS\\_LP\\_PassiveTamperPin\\_GetDefaultConfig](#) (snvs\_lp\_passive\_tamper\_t \*config)  
*Fills in the SNVS tamper pin config struct with the default settings.*

## Monotonic Counter (MC)

- static void [SNVS\\_LP\\_EnableMonotonicCounter](#) (SNVS\_Type \*base, bool enable)  
*Enable or disable the Monotonic Counter.*
- uint64\_t [SNVS\\_LP\\_GetMonotonicCounter](#) (SNVS\_Type \*base)  
*Get the current Monotonic Counter.*
- static void [SNVS\\_LP\\_IncreaseMonotonicCounter](#) (SNVS\_Type \*base)  
*Increase the Monotonic Counter.*

## Zeroizable Master Key (ZMK)

- void [SNVS\\_LP\\_WriteZeroizableMasterKey](#) (SNVS\_Type \*base, uint32\_t ZMKey[[SNVS\\_ZMK\\_REG\\_COUNT](#)])  
*Write Zeroizable Master Key (ZMK) to the SNVS registers.*
- static void [SNVS\\_LP\\_SetZeroizableMasterKeyValid](#) (SNVS\_Type \*base, bool valid)  
*Set Zeroizable Master Key valid.*
- static bool [SNVS\\_LP\\_GetZeroizableMasterKeyValid](#) (SNVS\_Type \*base)  
*Get Zeroizable Master Key valid status.*
- static void [SNVS\\_LP\\_SetZeroizableMasterKeyProgramMode](#) (SNVS\_Type \*base, [snvs\\_lp\\_zmk\\_program\\_mode\\_t](#) mode)  
*Set Zeroizable Master Key programming mode.*
- static void [SNVS\\_LP\\_EnableZeroizableMasterKeyECC](#) (SNVS\_Type \*base, bool enable)  
*Enable or disable Zeroizable Master Key ECC.*



- static void `SNVS_LP_SetMasterKeyMode` (`SNVS_Type *base`, `snvs_lp_master_key_mode_t mode`)  
*Set SNVS Master Key mode.*

## 41.3.2 Data Structure Documentation

### 41.3.2.1 struct `snvs_lp_passive_tamper_t`

### 41.3.2.2 struct `snvs_lp_srtc_datetime_t`

#### Data Fields

- `uint16_t year`  
*Range from 1970 to 2099.*
- `uint8_t month`  
*Range from 1 to 12.*
- `uint8_t day`  
*Range from 1 to 31 (depending on month).*
- `uint8_t hour`  
*Range from 0 to 23.*
- `uint8_t minute`  
*Range from 0 to 59.*
- `uint8_t second`  
*Range from 0 to 59.*

#### Field Documentation

- (1) `uint16_t snvs_lp_srtc_datetime_t::year`
- (2) `uint8_t snvs_lp_srtc_datetime_t::month`
- (3) `uint8_t snvs_lp_srtc_datetime_t::day`
- (4) `uint8_t snvs_lp_srtc_datetime_t::hour`
- (5) `uint8_t snvs_lp_srtc_datetime_t::minute`
- (6) `uint8_t snvs_lp_srtc_datetime_t::second`

### 41.3.2.3 struct `snvs_lp_srtc_config_t`

This structure holds the configuration settings for the SNVS\_LP peripheral. To initialize this structure to reasonable defaults, call the `SNVS_LP_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 `srtcCalEnable`  
*true: SRTC calibration mechanism is enabled; false: No calibration is used*
- uint32\_t `srtcCalValue`  
*Defines signed calibration value for SRTC; This is a 5-bit 2's complement value, range from -16 to +15.*

## 41.3.3 Enumeration Type Documentation

### 41.3.3.1 enum `snvs_lp_srtc_interrupts_t`

Enumerator

*kSNVS\_SRTC\_AlarmInterrupt* SRTC time alarm.

### 41.3.3.2 enum `snvs_lp_srtc_status_flags_t`

Enumerator

*kSNVS\_SRTC\_AlarmInterruptFlag* SRTC time alarm flag.

### 41.3.3.3 enum `snvs_lp_zmk_program_mode_t`

Enumerator

*kSNVS\_ZMKSoftwareProgram* Software programming mode.  
*kSNVS\_ZMKHardwareProgram* Hardware programming mode.

### 41.3.3.4 enum `snvs_lp_master_key_mode_t`

Enumerator

*kSNVS\_OTPMK* One Time Programmable Master Key.  
*kSNVS\_ZMK* Zeroizable Master Key.  
*kSNVS\_CMK* Combined Master Key, it is XOR of OPTMK and ZMK.

## 41.3.4 Function Documentation

### 41.3.4.1 void `SNVS_LP_Init ( SNVS_Type * base )`

Note

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

Parameters

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

#### 41.3.4.2 void SNVS\_LP\_Deinit ( SNVS\_Type \* *base* )

Parameters

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

#### 41.3.4.3 void SNVS\_LP\_SRTC\_Init ( SNVS\_Type \* *base*, const snvs\_lp\_srtc\_config\_t \* *config* )

Note

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

Parameters

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

#### 41.3.4.4 void SNVS\_LP\_SRTC\_Deinit ( SNVS\_Type \* *base* )

Parameters

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

#### 41.3.4.5 void SNVS\_LP\_SRTC\_GetDefaultConfig ( snvs\_lp\_srtc\_config\_t \* *config* )

The default values are as follows.

```
* config->srtccalenable = false;
* config->srtccalvalue = 0U;
*
```

## Parameters

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

#### 41.3.4.6 `status_t SNVS_LP_SRTC_SetDatetime ( SNVS_Type * base, const snvs_lp_srtc_datetime_t * datetime )`

## Parameters

|                 |                                                                      |
|-----------------|----------------------------------------------------------------------|
| <i>base</i>     | SNVS 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 SNVS SRTC  
`kStatus_InvalidArgument`: Error because the datetime format is incorrect

#### 41.3.4.7 `void SNVS_LP_SRTC_GetDatetime ( SNVS_Type * base, snvs_lp_srtc_datetime_t * datetime )`

## Parameters

|                 |                                                                      |
|-----------------|----------------------------------------------------------------------|
| <i>base</i>     | SNVS peripheral base address                                         |
| <i>datetime</i> | Pointer to the structure where the date and time details are stored. |

#### 41.3.4.8 `status_t SNVS_LP_SRTC_SetAlarm ( SNVS_Type * base, const snvs_lp_srtc_datetime_t * alarmTime )`

The function sets the SRTC alarm. It also 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. Please note, that SRTC alarm has limited resolution because only 32 most significant bits of SRTC counter are compared to SRTC Alarm register. If the alarm time is beyond SRTC resolution, the function does not set the alarm and returns an error.

## Parameters

|                  |                                                          |
|------------------|----------------------------------------------------------|
| <i>base</i>      | SNVS peripheral base address                             |
| <i>alarmTime</i> | Pointer to the structure where the alarm time is stored. |

## Returns

kStatus\_Success: success in setting the SNVS SRTC alarm  
kStatus\_InvalidArgument: Error because the alarm datetime format is incorrect  
kStatus\_Fail: Error because the alarm time has already passed or is beyond resolution

#### 41.3.4.9 void SNVS\_LP\_SRTC\_GetAlarm ( SNVS\_Type \* *base*, snvs\_lp\_srtc\_datetime\_t \* *datetime* )

## Parameters

|                 |                                                                            |
|-----------------|----------------------------------------------------------------------------|
| <i>base</i>     | SNVS peripheral base address                                               |
| <i>datetime</i> | Pointer to the structure where the alarm date and time details are stored. |

#### 41.3.4.10 static void SNVS\_LP\_SRTC\_EnableInterrupts ( SNVS\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                                                                       |
|-------------|-----------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SNVS peripheral base address                                                                                          |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration :: <code>_snvs_lp_srtc_interrupts</code> |

#### 41.3.4.11 static void SNVS\_LP\_SRTC\_DisableInterrupts ( SNVS\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                                                                       |
|-------------|-----------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SNVS peripheral base address                                                                                          |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration :: <code>_snvs_lp_srtc_interrupts</code> |

#### 41.3.4.12 uint32\_t SNVS\_LP\_SRTC\_GetEnabledInterrupts ( SNVS\_Type \* *base* )

## Parameters

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

## Returns

The enabled interrupts. This is the logical OR of members of the enumeration :: `_snvs_lp_srtc_`-`interrupts`

**41.3.4.13 `uint32_t SNVS_LP_SRTC_GetStatusFlags ( SNVS_Type * base )`**

## Parameters

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

## Returns

The status flags. This is the logical OR of members of the enumeration :: `_snvs_lp_srtc_status_flags`

**41.3.4.14 `static void SNVS_LP_SRTC_ClearStatusFlags ( SNVS_Type * base, uint32_t mask ) [inline], [static]`**

## Parameters

|             |                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SNVS peripheral base address                                                                                             |
| <i>mask</i> | The status flags to clear. This is a logical OR of members of the enumeration :: <code>_snvs_lp_srtc_status_flags</code> |

**41.3.4.15 `static void SNVS_LP_SRTC_StartTimer ( SNVS_Type * base ) [inline], [static]`**

## Parameters

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

**41.3.4.16 `static void SNVS_LP_SRTC_StopTimer ( SNVS_Type * base ) [inline], [static]`**

Parameters

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

**41.3.4.17 void SNVS\_LP\_PassiveTamperPin\_GetDefaultConfig ( snvs\_lp\_passive\_tamper\_t \* *config* )**

The default values are as follows. code config->polarity = 0U; config->filterenable = 0U; if available on SoC config->filter = 0U; if available on SoC endcode

Parameters

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

**41.3.4.18 static void SNVS\_LP\_EnableMonotonicCounter ( SNVS\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | SNVS peripheral base address           |
| <i>enable</i> | Pass true to enable, false to disable. |

**41.3.4.19 uint64\_t SNVS\_LP\_GetMonotonicCounter ( SNVS\_Type \* *base* )**

Parameters

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

Returns

Current Monotonic Counter value.

**41.3.4.20 static void SNVS\_LP\_IncreaseMonotonicCounter ( SNVS\_Type \* *base* ) [inline], [static]**

Increase the Monotonic Counter by 1.

Parameters

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

**41.3.4.21 void SNVS\_LP\_WriteZeroizableMasterKey ( SNVS\_Type \* *base*, uint32\_t *ZMKey*[SNVS\_ZMK\_REG\_COUNT] )**

Parameters

|              |                                     |
|--------------|-------------------------------------|
| <i>base</i>  | SNVS peripheral base address        |
| <i>ZMKey</i> | The ZMK write to the SNVS register. |

**41.3.4.22 static void SNVS\_LP\_SetZeroizableMasterKeyValid ( SNVS\_Type \* *base*, bool *valid* ) [inline], [static]**

This API could only be called when using software programming mode. After writing ZMK using [SNVS\\_LP\\_WriteZeroizableMasterKey](#), call this API to make the ZMK valid.

Parameters

|              |                                               |
|--------------|-----------------------------------------------|
| <i>base</i>  | SNVS peripheral base address                  |
| <i>valid</i> | Pass true to set valid, false to set invalid. |

**41.3.4.23 static bool SNVS\_LP\_GetZeroizableMasterKeyValid ( SNVS\_Type \* *base* ) [inline], [static]**

In hardware programming mode, call this API to check whether the ZMK is valid.

Parameters

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

Returns

true if valid, false if invalid.

**41.3.4.24 static void SNVS\_LP\_SetZeroizableMasterKeyProgramMode ( SNVS\_Type \* *base*, snvs\_lp\_zmk\_program\_mode\_t *mode* ) [inline], [static]**



Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | SNVS peripheral base address |
| <i>mode</i> | ZMK programming mode.        |

**41.3.4.25** `static void SNVS_LP_EnableZeroizableMasterKeyECC ( SNVS_Type * base,  
bool enable ) [inline], [static]`

Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | SNVS peripheral base address           |
| <i>enable</i> | Pass true to enable, false to disable. |

**41.3.4.26** `static void SNVS_LP_SetMasterKeyMode ( SNVS_Type * base,  
snvs_lp_master_key_mode_t mode ) [inline], [static]`

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | SNVS peripheral base address |
| <i>mode</i> | Master Key mode.             |

Note

When [kSNVS\\_ZMK](#) or [kSNVS\\_CMK](#) used, the SNVS\_HP must be configured to enable the master key selection.

## Chapter 42

# SPDIF: Sony/Philips Digital Interface

### 42.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Sony/Philips Digital Interface (SPDIF) module of MCUXpresso SDK devices.

SPDIF driver includes functional APIs and transactional APIs.

Functional APIs target low-level APIs. Functional APIs can be used for SPDIF initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the SPDIF peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SPDIF 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 `spdif_handle_t` as the first parameter. Initialize the handle by calling the [SPDIF\\_TransferTxCreateHandle\(\)](#) or [SPDIF\\_TransferRxCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [SPDIF\\_TransferSendNonBlocking\(\)](#) and [SPDIF\\_TransferReceiveNonBlocking\(\)](#) set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_SPDIF_TxIdle` and `kStatus_SPDIF_RxIdle` status.

### 42.2 Typical use case

#### 42.2.1 SPDIF Send/receive using an interrupt method

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

#### 42.2.2 SPDIF Send/receive using a DMA method

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

### Modules

- [SPDIF eDMA Driver](#)

### Data Structures

- struct [spdif\\_config\\_t](#)

- *SPDIF user configuration structure. [More...](#)*
- struct `spdif_transfer_t`  
*SPDIF transfer structure. [More...](#)*
- struct `spdif_handle_t`  
*SPDIF handle structure. [More...](#)*

## Macros

- #define `SPDIF_XFER_QUEUE_SIZE` (4U)  
*SPDIF transfer queue size, user can refine it according to use case.*

## Typedefs

- typedef void(\* `spdif_transfer_callback_t`)(SPDIF\_Type \*base, spdif\_handle\_t \*handle, `status_t` status, void \*userData)  
*SPDIF transfer callback prototype.*

## Enumerations

- enum {  
`kStatus_SPDIF_RxDPLLLocked` = MAKE\_STATUS(kStatusGroup\_SPDIF, 0),  
`kStatus_SPDIF_TxFIFOError` = MAKE\_STATUS(kStatusGroup\_SPDIF, 1),  
`kStatus_SPDIF_TxFIFOResync` = MAKE\_STATUS(kStatusGroup\_SPDIF, 2),  
`kStatus_SPDIF_RxCnew` = MAKE\_STATUS(kStatusGroup\_SPDIF, 3),  
`kStatus_SPDIF_ValidatyNoGood` = MAKE\_STATUS(kStatusGroup\_SPDIF, 4),  
`kStatus_SPDIF_RxIllegalSymbol` = MAKE\_STATUS(kStatusGroup\_SPDIF, 5),  
`kStatus_SPDIF_RxParityBitError` = MAKE\_STATUS(kStatusGroup\_SPDIF, 6),  
`kStatus_SPDIF_UChannelOverrun` = MAKE\_STATUS(kStatusGroup\_SPDIF, 7),  
`kStatus_SPDIF_QChannelOverrun` = MAKE\_STATUS(kStatusGroup\_SPDIF, 8),  
`kStatus_SPDIF_UQChannelSync` = MAKE\_STATUS(kStatusGroup\_SPDIF, 9),  
`kStatus_SPDIF_UQChannelFrameError` = MAKE\_STATUS(kStatusGroup\_SPDIF, 10),  
`kStatus_SPDIF_RxFIFOError` = MAKE\_STATUS(kStatusGroup\_SPDIF, 11),  
`kStatus_SPDIF_RxFIFOResync` = MAKE\_STATUS(kStatusGroup\_SPDIF, 12),  
`kStatus_SPDIF_LockLoss` = MAKE\_STATUS(kStatusGroup\_SPDIF, 13),  
`kStatus_SPDIF_TxIdle` = MAKE\_STATUS(kStatusGroup\_SPDIF, 14),  
`kStatus_SPDIF_RxIdle` = MAKE\_STATUS(kStatusGroup\_SPDIF, 15),  
`kStatus_SPDIF_QueueFull` = MAKE\_STATUS(kStatusGroup\_SPDIF, 16) }  
*SPDIF return status.*
- enum `spdif_rxfull_select_t` {  
`kSPDIF_RxFull1Sample` = 0x0u,  
`kSPDIF_RxFull4Samples`,  
`kSPDIF_RxFull8Samples`,  
`kSPDIF_RxFull16Samples` }  
*SPDIF Rx FIFO full flag select, it decides when assert the rx full flag.*
- enum `spdif_txempty_select_t` {  
`kSPDIF_TxEmpty0Sample` = 0x0u,  
`kSPDIF_TxEmpty4Samples`,  
`kSPDIF_TxEmpty8Samples`,

- ```
kSPDIF_TxEmpty12Samples }
```
- SPDIF tx FIFO EMPTY flag select, it decides when assert the tx empty flag.*
- enum `spdif_uchannel_source_t` {


```
kSPDIF_NoUChannel = 0x0U,
kSPDIF_UChannelFromRx = 0x1U,
kSPDIF_UChannelFromTx = 0x3U }
```

SPDIF U channel source.
 - enum `spdif_gain_select_t` {


```
kSPDIF_GAIN_24 = 0x0U,
kSPDIF_GAIN_16,
kSPDIF_GAIN_12,
kSPDIF_GAIN_8,
kSPDIF_GAIN_6,
kSPDIF_GAIN_4,
kSPDIF_GAIN_3 }
```

SPDIF clock gain.
 - enum `spdif_tx_source_t` {


```
kSPDIF_txFromReceiver = 0x1U,
kSPDIF_txNormal = 0x5U }
```

SPDIF tx data source.
 - enum `spdif_validity_config_t` {


```
kSPDIF_validityFlagAlwaysSet = 0x0U,
kSPDIF_validityFlagAlwaysClear }
```

SPDIF tx data source.
 - enum {


```
kSPDIF_RxDPLLLocked = SPDIF_SIE_LOCK_MASK,
kSPDIF_TxFIFOError = SPDIF_SIE_TXUNOV_MASK,
kSPDIF_TxFIFOResync = SPDIF_SIE_TXRESYN_MASK,
kSPDIF_RxControlChannelChange = SPDIF_SIE_CNEW_MASK,
kSPDIF_VValidityFlagNoGood = SPDIF_SIE_VALNOGOOD_MASK,
kSPDIF_RxIllegalSymbol = SPDIF_SIE_SYMERR_MASK,
kSPDIF_RxParityBitError = SPDIF_SIE_BITERR_MASK,
kSPDIF_UChannelReceiveRegisterFull = SPDIF_SIE_URXFUL_MASK,
kSPDIF_UChannelReceiveRegisterOverrun = SPDIF_SIE_URXOV_MASK,
kSPDIF_QChannelReceiveRegisterFull = SPDIF_SIE_QRXFUL_MASK,
kSPDIF_QChannelReceiveRegisterOverrun = SPDIF_SIE_QRXOV_MASK,
kSPDIF_UQChannelSync = SPDIF_SIE_UQSYNC_MASK,
kSPDIF_UQChannelFrameError = SPDIF_SIE_UQERR_MASK,
kSPDIF_RxFIFOError = SPDIF_SIE_RXFIFOUNOV_MASK,
kSPDIF_RxFIFOResync = SPDIF_SIE_RXFIFORESYN_MASK,
kSPDIF_LockLoss = SPDIF_SIE_LOCKLOSS_MASK,
kSPDIF_TxFIFOEmpty = SPDIF_SIE_TXEM_MASK,
kSPDIF_RxFIFOFull = SPDIF_SIE_RXFIFOFUL_MASK,
kSPDIF_AllInterrupt }
```

The SPDIF interrupt enable flag.
 - enum {

```
kSPDIF_RxDMAEnable = SPDIF_SCR_DMA_RX_EN_MASK,
kSPDIF_TxDMAEnable = SPDIF_SCR_DMA_TX_EN_MASK }
```

The DMA request sources.

Driver version

- #define `FSL_SPDIF_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 6)`)
Version 2.0.6.

Initialization and deinitialization

- void `SPDIF_Init` (`SPDIF_Type *base`, const `spdif_config_t *config`)
Initializes the SPDIF peripheral.
- void `SPDIF_GetDefaultConfig` (`spdif_config_t *config`)
Sets the SPDIF configuration structure to default values.
- void `SPDIF_Deinit` (`SPDIF_Type *base`)
De-initializes the SPDIF peripheral.
- uint32_t `SPDIF_GetInstance` (`SPDIF_Type *base`)
Get the instance number for SPDIF.
- static void `SPDIF_TxFIFOReset` (`SPDIF_Type *base`)
Resets the SPDIF Tx.
- static void `SPDIF_RxFIFOReset` (`SPDIF_Type *base`)
Resets the SPDIF Rx.
- void `SPDIF_TxEnable` (`SPDIF_Type *base`, bool enable)
Enables/disables the SPDIF Tx.
- static void `SPDIF_RxEnable` (`SPDIF_Type *base`, bool enable)
Enables/disables the SPDIF Rx.

Status

- static uint32_t `SPDIF_GetStatusFlag` (`SPDIF_Type *base`)
Gets the SPDIF status flag state.
- static void `SPDIF_ClearStatusFlags` (`SPDIF_Type *base`, uint32_t mask)
Clears the SPDIF status flag state.

Interrupts

- static void `SPDIF_EnableInterrupts` (`SPDIF_Type *base`, uint32_t mask)
Enables the SPDIF Tx interrupt requests.
- static void `SPDIF_DisableInterrupts` (`SPDIF_Type *base`, uint32_t mask)
Disables the SPDIF Tx interrupt requests.

DMA Control

- static void `SPDIF_EnableDMA` (`SPDIF_Type *base`, uint32_t mask, bool enable)
Enables/disables the SPDIF DMA requests.
- static uint32_t `SPDIF_TxGetLeftDataRegisterAddress` (`SPDIF_Type *base`)
Gets the SPDIF Tx left data register address.
- static uint32_t `SPDIF_TxGetRightDataRegisterAddress` (`SPDIF_Type *base`)
Gets the SPDIF Tx right data register address.

- static uint32_t [SPDIF_RxGetLeftDataRegisterAddress](#) (SPDIF_Type *base)
Gets the SPDIF Rx left data register address.
- static uint32_t [SPDIF_RxGetRightDataRegisterAddress](#) (SPDIF_Type *base)
Gets the SPDIF Rx right data register address.

Bus Operations

- void [SPDIF_TxSetSampleRate](#) (SPDIF_Type *base, uint32_t sampleRate_Hz, uint32_t sourceClockFreq_Hz)
Configures the SPDIF Tx sample rate.
- uint32_t [SPDIF_GetRxSampleRate](#) (SPDIF_Type *base, uint32_t clockSourceFreq_Hz)
Configures the SPDIF Rx audio format.
- void [SPDIF_WriteBlocking](#) (SPDIF_Type *base, uint8_t *buffer, uint32_t size)
Sends data using a blocking method.
- static void [SPDIF_WriteLeftData](#) (SPDIF_Type *base, uint32_t data)
Writes data into SPDIF FIFO.
- static void [SPDIF_WriteRightData](#) (SPDIF_Type *base, uint32_t data)
Writes data into SPDIF FIFO.
- static void [SPDIF_WriteChannelStatusHigh](#) (SPDIF_Type *base, uint32_t data)
Writes data into SPDIF FIFO.
- static void [SPDIF_WriteChannelStatusLow](#) (SPDIF_Type *base, uint32_t data)
Writes data into SPDIF FIFO.
- void [SPDIF_ReadBlocking](#) (SPDIF_Type *base, uint8_t *buffer, uint32_t size)
Receives data using a blocking method.
- static uint32_t [SPDIF_ReadLeftData](#) (SPDIF_Type *base)
Reads data from the SPDIF FIFO.
- static uint32_t [SPDIF_ReadRightData](#) (SPDIF_Type *base)
Reads data from the SPDIF FIFO.
- static uint32_t [SPDIF_ReadChannelStatusHigh](#) (SPDIF_Type *base)
Reads data from the SPDIF FIFO.
- static uint32_t [SPDIF_ReadChannelStatusLow](#) (SPDIF_Type *base)
Reads data from the SPDIF FIFO.
- static uint32_t [SPDIF_ReadQChannel](#) (SPDIF_Type *base)
Reads data from the SPDIF FIFO.
- static uint32_t [SPDIF_ReadUChannel](#) (SPDIF_Type *base)
Reads data from the SPDIF FIFO.

Transactional

- void [SPDIF_TransferTxCreateHandle](#) (SPDIF_Type *base, spdif_handle_t *handle, [spdif_transfer_callback_t](#) callback, void *userData)
Initializes the SPDIF Tx handle.
- void [SPDIF_TransferRxCreateHandle](#) (SPDIF_Type *base, spdif_handle_t *handle, [spdif_transfer_callback_t](#) callback, void *userData)
Initializes the SPDIF Rx handle.
- status_t [SPDIF_TransferSendNonBlocking](#) (SPDIF_Type *base, spdif_handle_t *handle, [spdif_transfer_t](#) *xfer)
Performs an interrupt non-blocking send transfer on SPDIF.
- status_t [SPDIF_TransferReceiveNonBlocking](#) (SPDIF_Type *base, spdif_handle_t *handle, [spdif_transfer_t](#) *xfer)

- *Performs an interrupt non-blocking receive transfer on SPDIF.*
- [status_t SPDIF_TransferGetSendCount](#) (SPDIF_Type *base, spdif_handle_t *handle, size_t *count)
Gets a set byte count.
- [status_t SPDIF_TransferGetReceiveCount](#) (SPDIF_Type *base, spdif_handle_t *handle, size_t *count)
Gets a received byte count.
- [void SPDIF_TransferAbortSend](#) (SPDIF_Type *base, spdif_handle_t *handle)
Aborts the current send.
- [void SPDIF_TransferAbortReceive](#) (SPDIF_Type *base, spdif_handle_t *handle)
Aborts the current IRQ receive.
- [void SPDIF_TransferTxHandleIRQ](#) (SPDIF_Type *base, spdif_handle_t *handle)
Tx interrupt handler.
- [void SPDIF_TransferRxHandleIRQ](#) (SPDIF_Type *base, spdif_handle_t *handle)
Rx interrupt handler.

42.3 Data Structure Documentation

42.3.1 struct spdif_config_t

Data Fields

- [bool isTxAutoSync](#)
If auto sync mechanism open.
- [bool isRxAutoSync](#)
If auto sync mechanism open.
- [uint8_t DPLLClkSource](#)
SPDIF DPLL clock source, range from 0~15, meaning is chip-specific.
- [uint8_t txClkSource](#)
SPDIF tx clock source, range from 0~7, meaning is chip-specific.
- [spdif_rxfull_select_t rxFullSelect](#)
SPDIF rx buffer full select.
- [spdif_txempty_select_t txFullSelect](#)
SPDIF tx buffer empty select.
- [spdif_uchannel_source_t uChannelSrc](#)
U channel source.
- [spdif_tx_source_t txSource](#)
SPDIF tx data source.
- [spdif_validity_config_t validityConfig](#)
Validity flag config.
- [spdif_gain_select_t gain](#)
Rx receive clock measure gain parameter.

Field Documentation

(1) `spdif_gain_select_t spdif_config_t::gain`

42.3.2 struct spdif_transfer_t**Data Fields**

- `uint8_t * data`
Data start address to transfer.
- `uint8_t * qdata`
Data buffer for Q channel.
- `uint8_t * udata`
Data buffer for C channel.
- `size_t dataSize`
Transfer size.

Field Documentation

(1) `uint8_t* spdif_transfer_t::data`

(2) `size_t spdif_transfer_t::dataSize`

42.3.3 struct _spdif_handle**Data Fields**

- `uint32_t state`
Transfer status.
- `spdif_transfer_callback_t callback`
Callback function called at transfer event.
- `void * userData`
Callback parameter passed to callback function.
- `spdif_transfer_t spdifQueue [SPDIF_XFER_QUEUE_SIZE]`
Transfer queue storing queued transfer.
- `size_t transferSize [SPDIF_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.

42.4 Macro Definition Documentation**42.4.1 #define SPDIF_XFER_QUEUE_SIZE (4U)**

42.5 Enumeration Type Documentation

42.5.1 anonymous enum

Enumerator

kStatus_SPDIF_RxDPLLLocked SPDIF Rx PLL locked.
kStatus_SPDIF_TxFIFOError SPDIF Tx FIFO error.
kStatus_SPDIF_TxFIFOResync SPDIF Tx left and right FIFO resync.
kStatus_SPDIF_RxCnew SPDIF Rx status channel value updated.
kStatus_SPDIF_ValidatyNoGood SPDIF validaty flag not good.
kStatus_SPDIF_RxIllegalSymbol SPDIF Rx receive illegal symbol.
kStatus_SPDIF_RxParityBitError SPDIF Rx parity bit error.
kStatus_SPDIF_UChannelOverrun SPDIF receive U channel overrun.
kStatus_SPDIF_QChannelOverrun SPDIF receive Q channel overrun.
kStatus_SPDIF_UQChannelSync SPDIF U/Q channel sync found.
kStatus_SPDIF_UQChannelFrameError SPDIF U/Q channel frame error.
kStatus_SPDIF_RxFIFOError SPDIF Rx FIFO error.
kStatus_SPDIF_RxFIFOResync SPDIF Rx left and right FIFO resync.
kStatus_SPDIF_LockLoss SPDIF Rx PLL clock lock loss.
kStatus_SPDIF_TxIdle SPDIF Tx is idle.
kStatus_SPDIF_RxIdle SPDIF Rx is idle.
kStatus_SPDIF_QueueFull SPDIF queue full.

42.5.2 enum spdif_rxfull_select_t

Enumerator

kSPDIF_RxFull1Sample Rx full at least 1 sample in left and right FIFO.
kSPDIF_RxFull4Samples Rx full at least 4 sample in left and right FIFO.
kSPDIF_RxFull8Samples Rx full at least 8 sample in left and right FIFO.
kSPDIF_RxFull16Samples Rx full at least 16 sample in left and right FIFO.

42.5.3 enum spdif_txempty_select_t

Enumerator

kSPDIF_TxEmpty0Sample Tx empty at most 0 sample in left and right FIFO.
kSPDIF_TxEmpty4Samples Tx empty at most 4 sample in left and right FIFO.
kSPDIF_TxEmpty8Samples Tx empty at most 8 sample in left and right FIFO.
kSPDIF_TxEmpty12Samples Tx empty at most 12 sample in left and right FIFO.

42.5.4 enum spdif_uchannel_source_t

Enumerator

- kSPDIF_NoUChannel* No embedded U channel.
- kSPDIF_UChannelFromRx* U channel from receiver, it is CD mode.
- kSPDIF_UChannelFromTx* U channel from on chip tx.

42.5.5 enum spdif_gain_select_t

Enumerator

- kSPDIF_GAIN_24* Gain select is 24.
- kSPDIF_GAIN_16* Gain select is 16.
- kSPDIF_GAIN_12* Gain select is 12.
- kSPDIF_GAIN_8* Gain select is 8.
- kSPDIF_GAIN_6* Gain select is 6.
- kSPDIF_GAIN_4* Gain select is 4.
- kSPDIF_GAIN_3* Gain select is 3.

42.5.6 enum spdif_tx_source_t

Enumerator

- kSPDIF_txFromReceiver* Tx data directly through SPDIF receiver.
- kSPDIF_txNormal* Normal operation, data from processor.

42.5.7 enum spdif_validity_config_t

Enumerator

- kSPDIF_validityFlagAlwaysSet* Outgoing validity flags always set.
- kSPDIF_validityFlagAlwaysClear* Outgoing validity flags always clear.

42.5.8 anonymous enum

Enumerator

- kSPDIF_RxDPLLLocked* SPDIF DPLL locked.
- kSPDIF_TxFIFOError* Tx FIFO underrun or overrun.
- kSPDIF_TxFIFOResync* Tx FIFO left and right channel resync.

kSPDIF_RxControlChannelChange SPDIF Rx control channel value changed.
kSPDIF_ValidityFlagNoGood SPDIF validity flag no good.
kSPDIF_RxIllegalSymbol SPDIF receiver found illegal symbol.
kSPDIF_RxParityBitError SPDIF receiver found parity bit error.
kSPDIF_UChannelReceiveRegisterFull SPDIF U channel receive register full.
kSPDIF_UChannelReceiveRegisterOverrun SPDIF U channel receive register overrun.
kSPDIF_QChannelReceiveRegisterFull SPDIF Q channel receive register full.
kSPDIF_QChannelReceiveRegisterOverrun SPDIF Q channel receive register overrun.
kSPDIF_UQChannelSync SPDIF U/Q channel sync found.
kSPDIF_UQChannelFrameError SPDIF U/Q channel frame error.
kSPDIF_RxFIFOError SPDIF Rx FIFO underrun/overrun.
kSPDIF_RxFIFOResync SPDIF Rx left and right FIFO resync.
kSPDIF_LockLoss SPDIF receiver loss of lock.
kSPDIF_TxFIFOEmpty SPDIF Tx FIFO empty.
kSPDIF_RxFIFOFull SPDIF Rx FIFO full.
kSPDIF_AllInterrupt all interrupt

42.5.9 anonymous enum

Enumerator

kSPDIF_RxDMAEnable Rx FIFO full.
kSPDIF_TxDMAEnable Tx FIFO empty.

42.6 Function Documentation

42.6.1 void SPDIF_Init (SPDIF_Type * *base*, const spdif_config_t * *config*)

Un-gates the SPDIF clock, resets the module, and configures SPDIF with a configuration structure. The configuration structure can be custom filled or set with default values by [SPDIF_GetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the SPDIF driver. Otherwise, accessing the SPDIF module can cause a hard fault because the clock is not enabled.

Parameters

<i>base</i>	SPDIF base pointer
-------------	--------------------

<i>config</i>	SPDIF configuration structure.
---------------	--------------------------------

42.6.2 void SPDIF_GetDefaultConfig (spdif_config_t * config)

This API initializes the configuration structure for use in SPDIF_Init. The initialized structure can remain unchanged in SPDIF_Init, or it can be modified before calling SPDIF_Init. This is an example.

```
spdif_config_t config;
SPDIF_GetDefaultConfig(&config);
```

Parameters

<i>config</i>	pointer to master configuration structure
---------------	---

42.6.3 void SPDIF_Deinit (SPDIF_Type * base)

This API gates the SPDIF clock. The SPDIF module can't operate unless SPDIF_Init is called to enable the clock.

Parameters

<i>base</i>	SPDIF base pointer
-------------	--------------------

42.6.4 uint32_t SPDIF_GetInstance (SPDIF_Type * base)

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

42.6.5 static void SPDIF_TxFIFOReset (SPDIF_Type * base) [inline], [static]

This function makes Tx FIFO in reset mode.

Parameters

<i>base</i>	SPDIF base pointer
-------------	--------------------

42.6.6 static void SPDIF_RxFIFOReset (SPDIF_Type * *base*) [inline], [static]

This function enables the software reset and FIFO reset of SPDIF Rx. After reset, clear the reset bit.

Parameters

<i>base</i>	SPDIF base pointer
-------------	--------------------

42.6.7 void SPDIF_TxEnable (SPDIF_Type * *base*, bool *enable*)

Parameters

<i>base</i>	SPDIF base pointer
<i>enable</i>	True means enable SPDIF Tx, false means disable.

42.6.8 static void SPDIF_RxEnable (SPDIF_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	SPDIF base pointer
<i>enable</i>	True means enable SPDIF Rx, false means disable.

42.6.9 static uint32_t SPDIF_GetStatusFlag (SPDIF_Type * *base*) [inline], [static]

Parameters

<i>base</i>	SPDIF base pointer
-------------	--------------------

Returns

SPDIF status flag value. Use the `_spdif_interrupt_enable_t` to get the status value needed.

**42.6.10 static void SPDIF_ClearStatusFlags (SPDIF_Type * *base*, uint32_t *mask*)
[inline], [static]**

Parameters

<i>base</i>	SPDIF base pointer
<i>mask</i>	State mask. It can be a combination of the <code>_spdif_interrupt_enable_t</code> member. Notice these members cannot be included, as these flags cannot be cleared by writing 1 to these bits: <ul style="list-style-type: none"> • <code>kSPDIF_UChannelReceiveRegisterFull</code> • <code>kSPDIF_QChannelReceiveRegisterFull</code> • <code>kSPDIF_TxFIFOEmpty</code> • <code>kSPDIF_RxFIFOFull</code>

**42.6.11 static void SPDIF_EnableInterrupts (SPDIF_Type * *base*, uint32_t *mask*)
[inline], [static]**

Parameters

<i>base</i>	SPDIF 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"> • <code>kSPDIF_WordStartInterruptEnable</code> • <code>kSPDIF_SyncErrorInterruptEnable</code> • <code>kSPDIF_FIFOWarningInterruptEnable</code> • <code>kSPDIF_FIFORequestInterruptEnable</code> • <code>kSPDIF_FIFOErrorInterruptEnable</code>

**42.6.12 static void SPDIF_DisableInterrupts (SPDIF_Type * *base*, uint32_t *mask*)
[inline], [static]**

Parameters

<i>base</i>	SPDIF 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"> • kSPDIF_WordStartInterruptEnable • kSPDIF_SyncErrorInterruptEnable • kSPDIF_FIFOWarningInterruptEnable • kSPDIF_FIFORequestInterruptEnable • kSPDIF_FIFOErrorInterruptEnable

42.6.13 static void SPDIF_EnabledDMA (SPDIF_Type * *base*, uint32_t *mask*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	SPDIF base pointer
<i>mask</i>	SPDIF DMA enable mask, The parameter can be a combination of the following sources if defined <ul style="list-style-type: none"> • kSPDIF_RxDMAEnable • kSPDIF_TxDMAEnable
<i>enable</i>	True means enable DMA, false means disable DMA.

42.6.14 static uint32_t SPDIF_TxGetLeftDataRegisterAddress (SPDIF_Type * *base*) [inline], [static]

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

Returns

data register address.

42.6.15 `static uint32_t SPDIF_TxGetRightDataRegisterAddress (SPDIF_Type *
base) [inline], [static]`

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

Returns

data register address.

42.6.16 `static uint32_t SPDIF_RxGetLeftDataRegisterAddress (SPDIF_Type * base) [inline], [static]`

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

Returns

data register address.

42.6.17 `static uint32_t SPDIF_RxGetRightDataRegisterAddress (SPDIF_Type * base) [inline], [static]`

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

Returns

data register address.

42.6.18 `void SPDIF_TxSetSampleRate (SPDIF_Type * base, uint32_t sampleRate_Hz, uint32_t sourceClockFreq_Hz)`

The audio format can be changed at run-time. This function configures the sample rate.

Parameters

<i>base</i>	SPDIF base pointer.
<i>sampleRate_Hz</i>	SPDIF sample rate frequency in Hz.
<i>sourceClock-Freq_Hz</i>	SPDIF tx clock source frequency in Hz.

42.6.19 `uint32_t SPDIF_GetRxSampleRate (SPDIF_Type * base, uint32_t clockSourceFreq_Hz)`

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>	SPDIF base pointer.
<i>clockSource-Freq_Hz</i>	SPDIF system clock frequency in hz.

42.6.20 `void SPDIF_WriteBlocking (SPDIF_Type * base, uint8_t * buffer, uint32_t size)`

Note

This function blocks by polling until data is ready to be sent.

Parameters

<i>base</i>	SPDIF base pointer.
<i>buffer</i>	Pointer to the data to be written.
<i>size</i>	Bytes to be written.

42.6.21 `static void SPDIF_WriteLeftData (SPDIF_Type * base, uint32_t data)` `[inline], [static]`

Parameters

<i>base</i>	SPDIF base pointer.
<i>data</i>	Data needs to be written.

42.6.22 `static void SPDIF_WriteRightData (SPDIF_Type * base, uint32_t data)`
[inline], [static]

Parameters

<i>base</i>	SPDIF base pointer.
<i>data</i>	Data needs to be written.

42.6.23 `static void SPDIF_WriteChannelStatusHigh (SPDIF_Type * base, uint32_t`
***data*) [inline], [static]**

Parameters

<i>base</i>	SPDIF base pointer.
<i>data</i>	Data needs to be written.

42.6.24 `static void SPDIF_WriteChannelStatusLow (SPDIF_Type * base, uint32_t`
***data*) [inline], [static]**

Parameters

<i>base</i>	SPDIF base pointer.
<i>data</i>	Data needs to be written.

42.6.25 `void SPDIF_ReadBlocking (SPDIF_Type * base, uint8_t * buffer, uint32_t`
***size*)**

Note

This function blocks by polling until data is ready to be sent.

Parameters

<i>base</i>	SPDIF base pointer.
<i>buffer</i>	Pointer to the data to be read.
<i>size</i>	Bytes to be read.

42.6.26 `static uint32_t SPDIF_ReadLeftData (SPDIF_Type * base) [inline], [static]`

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

Returns

Data in SPDIF FIFO.

42.6.27 `static uint32_t SPDIF_ReadRightData (SPDIF_Type * base) [inline], [static]`

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

Returns

Data in SPDIF FIFO.

42.6.28 `static uint32_t SPDIF_ReadChannelStatusHigh (SPDIF_Type * base) [inline], [static]`

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

Returns

Data in SPDIF FIFO.

42.6.29 `static uint32_t SPDIF_ReadChannelStatusLow (SPDIF_Type * base)`
`[inline], [static]`

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

Returns

Data in SPDIF FIFO.

42.6.30 `static uint32_t SPDIF_ReadQChannel (SPDIF_Type * base)` `[inline],`
`[static]`

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

Returns

Data in SPDIF FIFO.

42.6.31 `static uint32_t SPDIF_ReadUChannel (SPDIF_Type * base)` `[inline],`
`[static]`

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

Returns

Data in SPDIF FIFO.

42.6.32 void SPDIF_TransferTxCreateHandle (SPDIF_Type * *base*, spdif_handle_t * *handle*, spdif_transfer_callback_t *callback*, void * *userData*)

This function initializes the Tx handle for the SPDIF Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

<i>base</i>	SPDIF base pointer
<i>handle</i>	SPDIF handle pointer.
<i>callback</i>	Pointer to the user callback function.
<i>userData</i>	User parameter passed to the callback function

42.6.33 void SPDIF_TransferRxCreateHandle (SPDIF_Type * *base*, spdif_handle_t * *handle*, spdif_transfer_callback_t *callback*, void * *userData*)

This function initializes the Rx handle for the SPDIF Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	SPDIF handle pointer.
<i>callback</i>	Pointer to the user callback function.
<i>userData</i>	User parameter passed to the callback function.

42.6.34 status_t SPDIF_TransferSendNonBlocking (SPDIF_Type * *base*, spdif_handle_t * *handle*, spdif_transfer_t * *xfer*)

Note

This API returns immediately after the transfer initiates. Call the SPDIF_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SPDIF_Busy, the transfer is finished.

Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	Pointer to the spdif_handle_t structure which stores the transfer state.
<i>xfer</i>	Pointer to the spdif_transfer_t structure.

Return values

<i>kStatus_Success</i>	Successfully started the data receive.
<i>kStatus_SPDIF_TxBusy</i>	Previous receive still not finished.
<i>kStatus_InvalidArgument</i>	The input parameter is invalid.

42.6.35 `status_t SPDIF_TransferReceiveNonBlocking (SPDIF_Type * base, spdif_handle_t * handle, spdif_transfer_t * xfer)`

Note

This API returns immediately after the transfer initiates. Call the `SPDIF_RxGetTransferStatusIRQ` to poll the transfer status and check whether the transfer is finished. If the return status is not `kStatus_SPDIF_Busy`, the transfer is finished.

Parameters

<i>base</i>	SPDIF base pointer
<i>handle</i>	Pointer to the <code>spdif_handle_t</code> structure which stores the transfer state.
<i>xfer</i>	Pointer to the <code>spdif_transfer_t</code> structure.

Return values

<i>kStatus_Success</i>	Successfully started the data receive.
<i>kStatus_SPDIF_RxBusy</i>	Previous receive still not finished.
<i>kStatus_InvalidArgument</i>	The input parameter is invalid.

42.6.36 `status_t SPDIF_TransferGetSendCount (SPDIF_Type * base, spdif_handle_t * handle, size_t * count)`

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

<i>handle</i>	Pointer to the <code>spdif_handle_t</code> 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.

42.6.37 `status_t SPDIF_TransferGetReceiveCount (SPDIF_Type * base, spdif_handle_t * handle, size_t * count)`

Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	Pointer to the <code>spdif_handle_t</code> 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.

42.6.38 `void SPDIF_TransferAbortSend (SPDIF_Type * base, spdif_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>	SPDIF base pointer.
<i>handle</i>	Pointer to the <code>spdif_handle_t</code> structure which stores the transfer state.

42.6.39 void SPDIF_TransferAbortReceive (SPDIF_Type * *base*, spdif_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>	SPDIF base pointer
<i>handle</i>	Pointer to the spdif_handle_t structure which stores the transfer state.

42.6.40 void SPDIF_TransferTxHandleIRQ (SPDIF_Type * *base*, spdif_handle_t * *handle*)

Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	Pointer to the spdif_handle_t structure.

42.6.41 void SPDIF_TransferRxHandleIRQ (SPDIF_Type * *base*, spdif_handle_t * *handle*)

Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	Pointer to the spdif_handle_t structure.

42.7 SPDIF eDMA Driver

42.7.1 Overview

Data Structures

- struct `spdif_edma_transfer_t`
SPDIF transfer structure. [More...](#)
- struct `spdif_edma_handle_t`
SPDIF DMA transfer handle, users should not touch the content of the handle. [More...](#)

Typedefs

- typedef `void(* spdif_edma_callback_t)(SPDIF_Type *base, spdif_edma_handle_t *handle, status_t status, void *userData)`
SPDIF eDMA transfer callback function for finish and error.

Driver version

- `#define FSL_SPDIF_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))`
Version 2.0.5.

eDMA Transactional

- void `SPDIF_TransferTxCreateHandleEDMA` (SPDIF_Type *base, spdif_edma_handle_t *handle, `spdif_edma_callback_t` callback, void *userData, `edma_handle_t` *dmaLeftHandle, `edma_handle_t` *dmaRightHandle)
Initializes the SPDIF eDMA handle.
- void `SPDIF_TransferRxCreateHandleEDMA` (SPDIF_Type *base, spdif_edma_handle_t *handle, `spdif_edma_callback_t` callback, void *userData, `edma_handle_t` *dmaLeftHandle, `edma_handle_t` *dmaRightHandle)
Initializes the SPDIF Rx eDMA handle.
- `status_t SPDIF_TransferSendEDMA` (SPDIF_Type *base, spdif_edma_handle_t *handle, `spdif_edma_transfer_t` *xfer)
Performs a non-blocking SPDIF transfer using DMA.
- `status_t SPDIF_TransferReceiveEDMA` (SPDIF_Type *base, spdif_edma_handle_t *handle, `spdif_edma_transfer_t` *xfer)
Performs a non-blocking SPDIF receive using eDMA.
- void `SPDIF_TransferAbortSendEDMA` (SPDIF_Type *base, spdif_edma_handle_t *handle)
Aborts a SPDIF transfer using eDMA.
- void `SPDIF_TransferAbortReceiveEDMA` (SPDIF_Type *base, spdif_edma_handle_t *handle)
Aborts a SPDIF receive using eDMA.
- `status_t SPDIF_TransferGetSendCountEDMA` (SPDIF_Type *base, spdif_edma_handle_t *handle, `size_t` *count)
Gets byte count sent by SPDIF.

- `status_t SPDIF_TransferGetReceiveCountEDMA` (`SPDIF_Type *base, spdif_edma_handle_t *handle, size_t *count`)
Gets byte count received by SPDIF.

42.7.2 Data Structure Documentation

42.7.2.1 struct `spdif_edma_transfer_t`

Data Fields

- `uint8_t * leftData`
Data start address to transfer.
- `uint8_t * rightData`
Data start address to transfer.
- `size_t dataSize`
Transfer size.

Field Documentation

- (1) `uint8_t* spdif_edma_transfer_t::leftData`
- (2) `uint8_t* spdif_edma_transfer_t::rightData`
- (3) `size_t spdif_edma_transfer_t::dataSize`

42.7.2.2 struct `_spdif_edma_handle`

Data Fields

- `edma_handle_t * dmaLeftHandle`
DMA handler for SPDIF left channel.
- `edma_handle_t * dmaRightHandle`
DMA handler for SPDIF right channel.
- `uint8_t nbytes`
eDMA minor byte transfer count initially configured.
- `uint8_t count`
The transfer data count in a DMA request.
- `uint32_t state`
Internal state for SPDIF eDMA transfer.
- `spdif_edma_callback_t callback`
Callback for users while transfer finish or error occurs.
- `void * userData`
User callback parameter.
- `edma_tcd_t leftTcd [SPDIF_XFER_QUEUE_SIZE+1U]`
TCD pool for eDMA transfer.
- `edma_tcd_t rightTcd [SPDIF_XFER_QUEUE_SIZE+1U]`
TCD pool for eDMA transfer.
- `spdif_edma_transfer_t spdifQueue [SPDIF_XFER_QUEUE_SIZE]`
Transfer queue storing queued transfer.

- `size_t transferSize` [SPDIF_XFER_QUEUE_SIZE]
Data bytes need to transfer, left and right are the same, so use one.
- `volatile uint8_t queueUser`
Index for user to queue transfer.
- `volatile uint8_t queueDriver`
Index for driver to get the transfer data and size.

Field Documentation

- (1) `uint8_t spdif_edma_handle_t::nbytes`
- (2) `edma_tcd_t spdif_edma_handle_t::leftTcd[SPDIF_XFER_QUEUE_SIZE+1U]`
- (3) `edma_tcd_t spdif_edma_handle_t::rightTcd[SPDIF_XFER_QUEUE_SIZE+1U]`
- (4) `spdif_edma_transfer_t spdif_edma_handle_t::spdifQueue[SPDIF_XFER_QUEUE_SIZE]`
- (5) `volatile uint8_t spdif_edma_handle_t::queueUser`

42.7.3 Function Documentation

**42.7.3.1 void SPDIF_TransferTxCreateHandleEDMA (SPDIF_Type * *base*,
spdif_edma_handle_t * *handle*, spdif_edma_callback_t *callback*, void *
userData, edma_handle_t * *dmaLeftHandle*, edma_handle_t * *dmaRightHandle*)**

This function initializes the SPDIF master DMA handle, which can be used for other SPDIF master transactional APIs. Usually, for a specified SPDIF instance, call this API once to get the initialized handle.

Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	SPDIF eDMA handle pointer.
<i>base</i>	SPDIF peripheral base address.
<i>callback</i>	Pointer to user callback function.
<i>userData</i>	User parameter passed to the callback function.
<i>dmaLeftHandle</i>	eDMA handle pointer for left channel, this handle shall be static allocated by users.
<i>dmaRight-Handle</i>	eDMA handle pointer for right channel, this handle shall be static allocated by users.

**42.7.3.2 void SPDIF_TransferRxCreateHandleEDMA (SPDIF_Type * *base*,
spdif_edma_handle_t * *handle*, spdif_edma_callback_t *callback*, void *
userData, edma_handle_t * *dmaLeftHandle*, edma_handle_t * *dmaRightHandle*)**

This function initializes the SPDIF slave DMA handle, which can be used for other SPDIF master transactional APIs. Usually, for a specified SPDIF instance, call this API once to get the initialized handle.

Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	SPDIF eDMA handle pointer.
<i>base</i>	SPDIF peripheral base address.
<i>callback</i>	Pointer to user callback function.
<i>userData</i>	User parameter passed to the callback function.
<i>dmaLeftHandle</i>	eDMA handle pointer for left channel, this handle shall be static allocated by users.
<i>dmaRight-Handle</i>	eDMA handle pointer for right channel, this handle shall be static allocated by users.

42.7.3.3 `status_t SPDIF_TransferSendEDMA (SPDIF_Type * base, spdif_edma_handle_t * handle, spdif_edma_transfer_t * xfer)`

Note

This interface returns immediately after the transfer initiates. Call `SPDIF_GetTransferStatus` to poll the transfer status and check whether the SPDIF transfer is finished.

Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	SPDIF eDMA handle pointer.
<i>xfer</i>	Pointer to the DMA transfer structure.

Return values

<i>kStatus_Success</i>	Start a SPDIF eDMA send successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.
<i>kStatus_TxBusy</i>	SPDIF is busy sending data.

42.7.3.4 `status_t SPDIF_TransferReceiveEDMA (SPDIF_Type * base, spdif_edma_handle_t * handle, spdif_edma_transfer_t * xfer)`

Note

This interface returns immediately after the transfer initiates. Call the `SPDIF_GetReceive-RemainingBytes` to poll the transfer status and check whether the SPDIF transfer is finished.

Parameters

<i>base</i>	SPDIF base pointer
<i>handle</i>	SPDIF eDMA handle pointer.
<i>xfer</i>	Pointer to DMA transfer structure.

Return values

<i>kStatus_Success</i>	Start a SPDIF eDMA receive successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.
<i>kStatus_RxBusy</i>	SPDIF is busy receiving data.

42.7.3.5 void SPDIF_TransferAbortSendEDMA (SPDIF_Type * *base*, spdif_edma_handle_t * *handle*)

Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	SPDIF eDMA handle pointer.

42.7.3.6 void SPDIF_TransferAbortReceiveEDMA (SPDIF_Type * *base*, spdif_edma_handle_t * *handle*)

Parameters

<i>base</i>	SPDIF base pointer
<i>handle</i>	SPDIF eDMA handle pointer.

42.7.3.7 status_t SPDIF_TransferGetSendCountEDMA (SPDIF_Type * *base*, spdif_edma_handle_t * *handle*, size_t * *count*)

Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	SPDIF eDMA handle pointer.
<i>count</i>	Bytes count sent by SPDIF.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is no non-blocking transaction in progress.

42.7.3.8 **status_t SPDIF_TransferGetReceiveCountEDMA (SPDIF_Type * *base*, spdif_edma_handle_t * *handle*, size_t * *count*)**

Parameters

<i>base</i>	SPDIF base pointer
<i>handle</i>	SPDIF eDMA handle pointer.
<i>count</i>	Bytes count received by SPDIF.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is no non-blocking transaction in progress.

Chapter 43

SRC: System Reset Controller Driver

43.1 Overview

The MCUXpresso SDK provides a peripheral driver for the System Reset Controller (SRC) module.

The System Reset Controller (SRC) controls the reset and boot operation of the SoC. It is responsible for the generation of all reset signals and boot decoding. The reset controller determines the source and the type of reset, such as POR, WARM, COLD, and performs the necessary reset qualification and stretching sequences. Based on the type of reset, the reset logic generates the reset sequence for the entire IC.

Enumerations

- enum `_src_reset_status_flags` {
 `kSRC_WarmBootIndicationFlag` = SRC_SRSR_WBI_MASK,
 `kSRC_TemperatureSensorResetFlag` = SRC_SRSR_TSR_MASK,
 `kSRC_JTAGSoftwareResetFlag` = SRC_SRSR_SJC_MASK,
 `kSRC_JTAGGeneratedResetFlag` = SRC_SRSR_JTAG_MASK,
 `kSRC_WatchdogResetFlag` = SRC_SRSR_WDOG_MASK }
 SRC reset status flags.
- enum `src_warm_reset_bypass_count_t` {
 `kSRC_WarmResetWaitAlways` = 0U,
 `kSRC_WarmResetWaitClk16` = 1U,
 `kSRC_WarmResetWaitClk32` = 2U,
 `kSRC_WarmResetWaitClk64` = 3U }
 Selection of WARM reset bypass count.

Functions

- static void `SRC_EnableWDOGReset` (SRC_Type *base, bool enable)
 Enable the WDOG Reset in SRC.
- static void `SRC_SetWarmResetBypassCount` (SRC_Type *base, `src_warm_reset_bypass_count_t` option)
 Set the delay count of waiting MMDC's acknowledge.
- static void `SRC_EnableWarmReset` (SRC_Type *base, bool enable)
 Enable the WARM reset.
- static uint32_t `SRC_GetBootModeWord1` (SRC_Type *base)
 Get the boot mode register 1 value.
- static uint32_t `SRC_GetBootModeWord2` (SRC_Type *base)
 Get the boot mode register 2 value.
- static void `SRC_SetWarmBootIndication` (SRC_Type *base, bool enable)
 Set the warm boot indication flag.
- static uint32_t `SRC_GetResetStatusFlags` (SRC_Type *base)
 Get the status flags of SRC.

- void `SRC_ClearResetStatusFlags` (SRC_Type *base, uint32_t flags)
Clear the status flags of SRC.
- static void `SRC_SetGeneralPurposeRegister` (SRC_Type *base, uint32_t index, uint32_t value)
Set value to general purpose registers.
- static uint32_t `SRC_GetGeneralPurposeRegister` (SRC_Type *base, uint32_t index)
Get the value from general purpose registers.

Driver version

- #define `FSL_SRC_DRIVER_VERSION` (MAKE_VERSION(2, 0, 1))
SRC driver version 2.0.1.

43.2 Macro Definition Documentation

43.2.1 #define FSL_SRC_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

43.3 Enumeration Type Documentation

43.3.1 enum _src_reset_status_flags

Enumerator

kSRC_WarmBootIndicationFlag WARM boot indication shows that WARM boot was initiated by software.

kSRC_TemperatureSensorResetFlag Indicates whether the reset was the result of software reset from on-chip Temperature Sensor. Temperature Sensor Interrupt needs to be served before this bit can be cleaned.

kSRC_JTAGSoftwareResetFlag Indicates whether the reset was the result of setting SJC_GPCCR bit 31.

kSRC_JTAGGeneratedResetFlag Indicates a reset has been caused by JTAG selection of certain IR codes: EXTEST or HIGHZ.

kSRC_WatchdogResetFlag Indicates a reset has been caused by the watchdog timer timing out. This reset source can be blocked by disabling the watchdog.

43.3.2 enum src_warm_reset_bypass_count_t

This type defines the 32KHz clock cycles to count before bypassing the MMDC acknowledge for WARM reset. If the MMDC acknowledge is not asserted before this counter is elapsed, a COLD reset will be initiated.

Enumerator

kSRC_WarmResetWaitAlways System will wait until MMDC acknowledge is asserted.

kSRC_WarmResetWaitClk16 Wait 16 32KHz clock cycles before switching the reset.

kSRC_WarmResetWaitClk32 Wait 32 32KHz clock cycles before switching the reset.

kSRC_WarmResetWaitClk64 Wait 64 32KHz clock cycles before switching the reset.

43.4 Function Documentation

43.4.1 `static void SRC_EnableWDOGReset (SRC_Type * base, bool enable) [inline], [static]`

WDOG Reset is enabled in SRC by default. If the WDOG event to SRC is masked, it would not create a reset to the chip. During the time the WDOG event is masked, when the WDOG event flag is asserted, it would remain asserted regardless of servicing the WDOG module. The only way to clear that bit is the hardware reset.

Parameters

<i>base</i>	SRC peripheral base address.
<i>enable</i>	Enable the reset or not.

43.4.2 `static void SRC_SetWarmResetBypassCount (SRC_Type * base, src_warm_reset_bypass_count_t option) [inline], [static]`

This function would define the 32KHz clock cycles to count before bypassing the MMDC acknowledge for WARM reset. If the MMDC acknowledge is not asserted before this counter is elapsed, a COLD reset will be initiated.

Parameters

<i>base</i>	SRC peripheral base address.
<i>option</i>	The option of setting mode, see to src_warm_reset_bypass_count_t .

43.4.3 `static void SRC_EnableWarmReset (SRC_Type * base, bool enable) [inline], [static]`

Only when the WARM reset is enabled, the WARM reset requests would be served by WARM reset. Otherwise, all the WARM reset sources would generate COLD reset.

Parameters

<i>base</i>	SRC peripheral base address.
<i>enable</i>	Enable the WARM reset or not.

43.4.4 `static uint32_t SRC_GetBootModeWord1 (SRC_Type * base) [inline],
[static]`

The Boot Mode register contains bits that reflect the status of BOOT_CFGx pins of the chip. See to chip-specific document for detail information about value.

Parameters

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

Returns

status of BOOT_CFGx pins of the chip.

43.4.5 **static uint32_t SRC_GetBootModeWord2 (SRC_Type * *base*) [inline], [static]**

The Boot Mode register contains bits that reflect the status of BOOT_MODEx Pins and fuse values that controls boot of the chip. See to chip-specific document for detail information about value.

Parameters

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

Returns

status of BOOT_MODEx Pins and fuse values that controls boot of the chip.

43.4.6 **static void SRC_SetWarmBootIndication (SRC_Type * *base*, bool *enable*) [inline], [static]**

WARM boot indication shows that WARM boot was initiated by software. This indicates to the software that it saved the needed information in the memory before initiating the WARM reset. In this case, software will set this bit to '1', before initiating the WARM reset. The warm_boot bit should be used as indication only after a warm_reset sequence. Software should clear this bit after warm_reset to indicate that the next warm_reset is not performed with warm_boot.

Parameters

<i>base</i>	SRC peripheral base address.
<i>enable</i>	Assert the flag or not.

43.4.7 **static uint32_t SRC_GetResetStatusFlags (SRC_Type * *base*) [inline], [static]**

Parameters

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

Returns

Mask value of status flags, see to [_src_reset_status_flags](#).

43.4.8 void SRC_ClearResetStatusFlags (SRC_Type * *base*, uint32_t *flags*)

Parameters

<i>base</i>	SRC peripheral base address.
<i>flags</i>	value of status flags to be cleared, see to _src_reset_status_flags .

43.4.9 static void SRC_SetGeneralPurposeRegister (SRC_Type * *base*, uint32_t *index*, uint32_t *value*) [inline], [static]

General purpose registers (GPRx) would hold the value during reset process. Wakeup function could be kept in these register. For example, the GPR1 holds the entry function for waking-up from Partial SLEEP mode while the GPR2 holds the argument. Other GPRx register would store the arbitray values.

Parameters

<i>base</i>	SRC peripheral base address.
<i>index</i>	The index of GPRx register array. Note index 0 reponses the GPR1 register.
<i>value</i>	Setting value for GPRx register.

43.4.10 static uint32_t SRC_GetGeneralPurposeRegister (SRC_Type * *base*, uint32_t *index*) [inline], [static]

Parameters

<i>base</i>	SRC peripheral base address.
<i>index</i>	The index of GPRx register array. Note index 0 reponses the GPR1 register.

Returns

The setting value for GPRx register.

Chapter 44

TEMPMON: Temperature Monitor Module

44.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Temperature Monitor Module (TEMPMON) module of MCUXpresso SDK devices.

44.2 TEMPMON: Temperature Monitor Module

44.2.1 TEMPMON Operations

The function [TEMPMON_Init\(\)](#) will initialize the TEMPMON peripheral operation.

The function [TEMPMON_Deinit\(\)](#) will deinitialize the TEMPMON peripheral operation.

The function [TEMPMON_GetDefaultConfig\(\)](#) will get default configuration.

The function [TEMPMON_StartMeasure\(\)](#) will start the temperature measurement process.

The function [TEMPMON_StopMeasure\(\)](#) will stop the temperature measurement process.

The function [TEMPMON_GetCurrentTemp\(\)](#) will get the current temperature.

The function [TEMPMON_SetTempAlarm\(\)](#) will set the temperature count that will generate an alarm interrupt.

Files

- file [fsl_tempmon.h](#)

Data Structures

- struct [tempmon_config_t](#)
TEMPMON temperature structure. [More...](#)

Enumerations

- enum [tempmon_alarm_mode](#) {
 [kTEMPMON_HighAlarmMode](#) = 0U,
 [kTEMPMON_PanicAlarmMode](#) = 1U,
 [kTEMPMON_LowAlarmMode](#) = 2U }
TEMPMON alarm mode.

Functions

- void [TEMPMON_Init](#) (TEMPMON_Type *base, const [tempmon_config_t](#) *config)

- *Initializes the TEMPMON module.*
- void [TEMPMON_Deinit](#) (TEMPMON_Type *base)
Deinitializes the TEMPMON module.
- void [TEMPMON_GetDefaultConfig](#) ([tempmon_config_t](#) *config)
Gets the default configuration structure.
- static void [TEMPMON_StartMeasure](#) (TEMPMON_Type *base)
start the temperature measurement process.
- static void [TEMPMON_StopMeasure](#) (TEMPMON_Type *base)
stop the measurement process.
- float [TEMPMON_GetCurrentTemperature](#) (TEMPMON_Type *base)
Get current temperature with the fused temperature calibration data.
- void [TEMPMON_SetTempAlarm](#) (TEMPMON_Type *base, int8_t tempVal, [tempmon_alarm_mode](#) alarmMode)
Set the temperature count (raw sensor output) that will generate an alarm interrupt.

Driver version

- #define [FSL_TEMPMON_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 1, 1))
TEMPMON driver version.

44.3 Data Structure Documentation

44.3.1 struct tempmon_config_t

Data Fields

- uint16_t [frequency](#)
The temperature measure frequency.
- int8_t [highAlarmTemp](#)
The high alarm temperature.
- int8_t [panicAlarmTemp](#)
The panic alarm temperature.
- int8_t [lowAlarmTemp](#)
The low alarm temperature.

Field Documentation

- (1) [uint16_t tempmon_config_t::frequency](#)
- (2) [int8_t tempmon_config_t::highAlarmTemp](#)
- (3) [int8_t tempmon_config_t::panicAlarmTemp](#)
- (4) [int8_t tempmon_config_t::lowAlarmTemp](#)

44.4 Macro Definition Documentation

44.4.1 #define FSL_TEMPMON_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

44.5 Enumeration Type Documentation

44.5.1 enum tempmon_alarm_mode

Enumerator

- kTEMPMON_HighAlarmMode* The high alarm temperature interrupt mode.
- kTEMPMON_PanicAlarmMode* The panic alarm temperature interrupt mode.
- kTEMPMON_LowAlarmMode* The low alarm temperature interrupt mode.

44.6 Function Documentation

44.6.1 void TEMPMON_Init (TEMPMON_Type * *base*, const tempmon_config_t * *config*)

Parameters

<i>base</i>	TEMPMON base pointer
<i>config</i>	Pointer to configuration structure.

44.6.2 void TEMPMON_Deinit (TEMPMON_Type * *base*)

Parameters

<i>base</i>	TEMPMON base pointer
-------------	----------------------

44.6.3 void TEMPMON_GetDefaultConfig (tempmon_config_t * *config*)

This function initializes the TEMPMON configuration structure to a default value. The default values are: tempmonConfig->frequency = 0x02U; tempmonConfig->highAlarmTemp = 44U; tempmonConfig->panicAlarmTemp = 90U; tempmonConfig->lowAlarmTemp = 39U;

Parameters

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

44.6.4 static void TEMPMON_StartMeasure (TEMPMON_Type * *base*) [inline], [static]

Parameters

<i>base</i>	TEMPMON base pointer.
-------------	-----------------------

**44.6.5 static void TEMPMON_StopMeasure (TEMPMON_Type * *base*)
[inline], [static]**

Parameters

<i>base</i>	TEMPMON base pointer
-------------	----------------------

44.6.6 float TEMPMON_GetCurrentTemperature (TEMPMON_Type * *base*)

Parameters

<i>base</i>	TEMPMON base pointer
-------------	----------------------

Returns

current temperature with degrees Celsius.

**44.6.7 void TEMPMON_SetTempAlarm (TEMPMON_Type * *base*, int8_t *tempVal*,
tempmon_alarm_mode *alarmMode*)**

Parameters

<i>base</i>	TEMPMON base pointer
<i>tempVal</i>	The alarm temperature with degrees Celsius
<i>alarmMode</i>	The alarm mode.

Chapter 45

TRNG: True Random Number Generator

The MCUXpresso SDK provides a peripheral driver for the True Random Number Generator (TRNG) module of MCUXpresso SDK devices.

The True Random Number Generator is a hardware accelerator module that generates a 512-bit entropy as needed by an entropy consuming module or by other post processing functions. A typical entropy consumer is a pseudo random number generator (PRNG) which can be implemented to achieve both true randomness and cryptographic strength random numbers using the TRNG output as its entropy seed. The entropy generated by a TRNG is intended for direct use by functions that generate secret keys, per-message secrets, random challenges, and other similar quantities used in cryptographic algorithms.

45.1 TRNG Initialization

1. Define the TRNG user configuration structure. Use `TRNG_InitUserConfigDefault()` function to set it to default TRNG configuration values.
2. Initialize the TRNG module, call the `TRNG_Init()` function, and pass the user configuration structure. This function automatically enables the TRNG module and its clock. After that, the TRNG is enabled and the entropy generation starts working.
3. To disable the TRNG module, call the `TRNG_Deinit()` function.

45.2 Get random data from TRNG

1. `TRNG_GetRandomData()` function gets random data from the TRNG module.

This example code shows how to initialize and get random data from the TRNG driver.

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

Chapter 46

USDHC: Ultra Secured Digital Host Controller Driver

46.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Ultra Secured Digital Host Controller (USDHC) module of MCUXpresso SDK/i.MX devices.

46.2 Typical use case

46.2.1 USDHC Operation

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

Cache maintain capability

The uSDHC host controller is intergrated with ADMA to have better transfer performance, so to maintain data integrity during DMA operations on the platform that has cache, USDHC driver provide a cache maintain functionality by define: `FSL_SDK_ENABLE_DRIVER_CACHE_CONTROL = 1` It is suggest that the address of buffer used for read/write is align with cache line size.

Scatter gather transfer capability

The USDHC driver implement scatter gather transfer functionality, so application can submit uncontinuous data buffer in one transfer request by the scatter gather api, to have this feature, USDHC driver has below api `USDHC_TransferScatterGatherADMANonBlocking` This function support scatter gather transfer and cover the functionality of `USDHC_TransferNonBlocking` also, but if application would like to use the function, please enable function macro firstly, since the scatter gather functionality is disabled by default.

```
#define FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER 1
```

Please note that once the macro is defined, the `USDHC_TransferNonBlocking` will be removed automatically.

Data Structures

- struct `usdhc_adma2_descriptor_t`
Defines the ADMA2 descriptor structure. [More...](#)
- struct `usdhc_capability_t`
USDHC capability information. [More...](#)
- struct `usdhc_boot_config_t`
Data structure to configure the MMC boot feature. [More...](#)
- struct `usdhc_config_t`
Data structure to initialize the USDHC. [More...](#)
- struct `usdhc_command_t`
Card command descriptor. [More...](#)

- struct `usdhc_adma_config_t`
ADMA configuration. *More...*
- struct `usdhc_scatter_gather_data_list_t`
Card scatter gather data list. *More...*
- struct `usdhc_scatter_gather_data_t`
Card scatter gather data descriptor. *More...*
- struct `usdhc_scatter_gather_transfer_t`
usdhc scatter gather transfer. *More...*
- struct `usdhc_data_t`
Card data descriptor. *More...*
- struct `usdhc_transfer_t`
Transfer state. *More...*
- struct `usdhc_transfer_callback_t`
USDHC callback functions. *More...*
- struct `usdhc_handle_t`
USDHC handle. *More...*
- struct `usdhc_host_t`
USDHC host descriptor. *More...*

Macros

- #define `USDHC_MAX_BLOCK_COUNT` (`USDHC_BLK_ATT_BLKCNT_MASK >> USDHC-_BLK_ATT_BLKCNT_SHIFT`)
Maximum block count can be set one time.
- #define `FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER` 0U
USDHC scatter gather feature control macro.
- #define `USDHC_ADMA1_ADDRESS_ALIGN` (4096U)
The alignment size for ADDRESS filed in ADMA1's descriptor.
- #define `USDHC_ADMA1_LENGTH_ALIGN` (4096U)
The alignment size for LENGTH field in ADMA1's descriptor.
- #define `USDHC_ADMA2_ADDRESS_ALIGN` (4U)
The alignment size for ADDRESS field in ADMA2's descriptor.
- #define `USDHC_ADMA2_LENGTH_ALIGN` (4U)
The alignment size for LENGTH filed in ADMA2's descriptor.
- #define `USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT` (12U)
The bit shift for ADDRESS filed in ADMA1's descriptor.
- #define `USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK` (0xFFFFFU)
The bit mask for ADDRESS field in ADMA1's descriptor.
- #define `USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT` (12U)
The bit shift for LENGTH filed in ADMA1's descriptor.
- #define `USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK` (0xFFFFU)
The mask for LENGTH field in ADMA1's descriptor.
- #define `USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY` (`USDHC_ADMA1_-_DESCRIPTOR_LENGTH_MASK + 1U - 4096U`)
The maximum value of LENGTH filed in ADMA1's descriptor.
- #define `USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT` (16U)
The bit shift for LENGTH field in ADMA2's descriptor.
- #define `USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK` (0xFFFFFU)
The bit mask for LENGTH field in ADMA2's descriptor.
- #define `USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY` (`USDHC_ADMA2_-_DESCRIPTOR_LENGTH_MASK - 3U`)

The maximum value of *LENGTH* field in ADMA2's descriptor.

Typedefs

- typedef uint32_t `usdhc_adma1_descriptor_t`
Defines the ADMA1 descriptor structure.
- typedef `status_t`(* `usdhc_transfer_function_t`)(USDHC_Type *base, `usdhc_transfer_t` *content)
USDHC transfer function.

Enumerations

- enum {
`kStatus_USDHC_BusyTransferring` = MAKE_STATUS(kStatusGroup_USDHC, 0U),
`kStatus_USDHC_PrepareAdmaDescriptorFailed` = MAKE_STATUS(kStatusGroup_USDHC, 1U),
`kStatus_USDHC_SendCommandFailed` = MAKE_STATUS(kStatusGroup_USDHC, 2U),
`kStatus_USDHC_TransferDataFailed` = MAKE_STATUS(kStatusGroup_USDHC, 3U),
`kStatus_USDHC_DMADDataAddrNotAlign` = MAKE_STATUS(kStatusGroup_USDHC, 4U),
`kStatus_USDHC_ReTuningRequest` = MAKE_STATUS(kStatusGroup_USDHC, 5U),
`kStatus_USDHC_TuningError` = MAKE_STATUS(kStatusGroup_USDHC, 6U),
`kStatus_USDHC_NotSupport` = MAKE_STATUS(kStatusGroup_USDHC, 7U),
`kStatus_USDHC_TransferDataComplete` = MAKE_STATUS(kStatusGroup_USDHC, 8U),
`kStatus_USDHC_SendCommandSuccess` = MAKE_STATUS(kStatusGroup_USDHC, 9U),
`kStatus_USDHC_TransferDMACComplete` = MAKE_STATUS(kStatusGroup_USDHC, 10U) }
Enum _usdhc_status.
- enum {
`kUSDHC_SupportAdmaFlag` = USDHC_HOST_CTRL_CAP_ADMAS_MASK,
`kUSDHC_SupportHighSpeedFlag` = USDHC_HOST_CTRL_CAP_HSS_MASK,
`kUSDHC_SupportDmaFlag` = USDHC_HOST_CTRL_CAP_DMAS_MASK,
`kUSDHC_SupportSuspendResumeFlag` = USDHC_HOST_CTRL_CAP_SRS_MASK,
`kUSDHC_SupportV330Flag` = USDHC_HOST_CTRL_CAP_VS33_MASK,
`kUSDHC_SupportV300Flag` = USDHC_HOST_CTRL_CAP_VS30_MASK,
`kUSDHC_SupportV180Flag` = USDHC_HOST_CTRL_CAP_VS18_MASK,
`kUSDHC_Support4BitFlag` = (USDHC_HOST_CTRL_CAP_MBL_SHIFT << 0U),
`kUSDHC_Support8BitFlag` = (USDHC_HOST_CTRL_CAP_MBL_SHIFT << 1U),
`kUSDHC_SupportDDR50Flag` = USDHC_HOST_CTRL_CAP_DDR50_SUPPORT_MASK,
`kUSDHC_SupportSDR104Flag` = USDHC_HOST_CTRL_CAP_SDR104_SUPPORT_MASK,
`kUSDHC_SupportSDR50Flag` = USDHC_HOST_CTRL_CAP_SDR50_SUPPORT_MASK }
Enum _usdhc_capability_flag.
- enum {
`kUSDHC_WakeupEventOnCardInt` = USDHC_PROT_CTRL_WECINT_MASK,
`kUSDHC_WakeupEventOnCardInsert` = USDHC_PROT_CTRL_WECINS_MASK,
`kUSDHC_WakeupEventOnCardRemove` = USDHC_PROT_CTRL_WECRM_MASK,
`kUSDHC_WakeupEventsAll` }
Enum _usdhc_wakeup_event.
- enum {


```

kUSDHC_ResetAll = USDHC_SYS_CTRL_RSTA_MASK,
kUSDHC_ResetCommand = USDHC_SYS_CTRL_RSTC_MASK,
kUSDHC_ResetData = USDHC_SYS_CTRL_RSTD_MASK,
kUSDHC_ResetTuning = USDHC_SYS_CTRL_RSTT_MASK,
kUSDHC_ResetsAll = (kUSDHC_ResetAll | kUSDHC_ResetCommand | kUSDHC_ResetData |
kUSDHC_ResetTuning) }

```

Enum _usdhc_reset.

- enum {


```

kUSDHC_EnableDmaFlag = USDHC_MIX_CTRL_DMAEN_MASK,
kUSDHC_CommandTypeSuspendFlag = USDHC_CMD_XFR_TYP_CMDTYP(1U),
kUSDHC_CommandTypeResumeFlag = USDHC_CMD_XFR_TYP_CMDTYP(2U),
kUSDHC_CommandTypeAbortFlag = USDHC_CMD_XFR_TYP_CMDTYP(3U),
kUSDHC_EnableBlockCountFlag = USDHC_MIX_CTRL_BCEN_MASK,
kUSDHC_EnableAutoCommand12Flag = USDHC_MIX_CTRL_AC12EN_MASK,
kUSDHC_DataReadFlag = USDHC_MIX_CTRL_DTDSEL_MASK,
kUSDHC_MultipleBlockFlag = USDHC_MIX_CTRL_MSBSSEL_MASK,
kUSDHC_EnableAutoCommand23Flag = USDHC_MIX_CTRL_AC23EN_MASK,
kUSDHC_ResponseLength136Flag = USDHC_CMD_XFR_TYP_RSPTYP(1U),
kUSDHC_ResponseLength48Flag = USDHC_CMD_XFR_TYP_RSPTYP(2U),
kUSDHC_ResponseLength48BusyFlag = USDHC_CMD_XFR_TYP_RSPTYP(3U),
kUSDHC_EnableCrcCheckFlag = USDHC_CMD_XFR_TYP_CCCEN_MASK,
kUSDHC_EnableIndexCheckFlag = USDHC_CMD_XFR_TYP_CICEN_MASK,
kUSDHC_DataPresentFlag = USDHC_CMD_XFR_TYP_DPSEL_MASK }

```

Enum _usdhc_transfer_flag.

- enum {


```

kUSDHC_CommandInhibitFlag = USDHC_PRES_STATE_CIHB_MASK,
kUSDHC_DataInhibitFlag = USDHC_PRES_STATE_CDIHB_MASK,
kUSDHC_DataLineActiveFlag = USDHC_PRES_STATE_DLA_MASK,
kUSDHC_SdClockStableFlag = USDHC_PRES_STATE_SDSTB_MASK,
kUSDHC_WriteTransferActiveFlag = USDHC_PRES_STATE_WTA_MASK,
kUSDHC_ReadTransferActiveFlag = USDHC_PRES_STATE_RTA_MASK,
kUSDHC_BufferWriteEnableFlag = USDHC_PRES_STATE_BWEN_MASK,
kUSDHC_BufferReadEnableFlag = USDHC_PRES_STATE_BREN_MASK,
kUSDHC_ReTuningRequestFlag = USDHC_PRES_STATE_RTR_MASK,
kUSDHC_DelaySettingFinishedFlag = USDHC_PRES_STATE_TSCD_MASK,
kUSDHC_CardInsertedFlag = USDHC_PRES_STATE_CINST_MASK,
kUSDHC_CommandLineLevelFlag = USDHC_PRES_STATE_CLSL_MASK,
kUSDHC_Data0LineLevelFlag = 1U << USDHC_PRES_STATE_DLSSL_SHIFT,
kUSDHC_Data1LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 1U),
kUSDHC_Data2LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 2U),
kUSDHC_Data3LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 3U),
kUSDHC_Data4LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 4U),
kUSDHC_Data5LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 5U),
kUSDHC_Data6LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 6U),
kUSDHC_Data7LineLevelFlag = (int)(1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 7U)) }

```

- Enum *_usdhc_present_status_flag*.
 - enum {
 - kUSDHC_CommandCompleteFlag = USDHC_INT_STATUS_CC_MASK,
 - kUSDHC_DataCompleteFlag = USDHC_INT_STATUS_TC_MASK,
 - kUSDHC_BlockGapEventFlag = USDHC_INT_STATUS_BGE_MASK,
 - kUSDHC_DmaCompleteFlag = USDHC_INT_STATUS_DINT_MASK,
 - kUSDHC_BufferWriteReadyFlag = USDHC_INT_STATUS_BWR_MASK,
 - kUSDHC_BufferReadReadyFlag = USDHC_INT_STATUS_BRR_MASK,
 - kUSDHC_CardInsertionFlag = USDHC_INT_STATUS_CINS_MASK,
 - kUSDHC_CardRemovalFlag = USDHC_INT_STATUS_CRM_MASK,
 - kUSDHC_CardInterruptFlag = USDHC_INT_STATUS_CINT_MASK,
 - kUSDHC_ReTuningEventFlag = USDHC_INT_STATUS_RTE_MASK,
 - kUSDHC_TuningPassFlag = USDHC_INT_STATUS_TP_MASK,
 - kUSDHC_TuningErrorFlag = USDHC_INT_STATUS_TNE_MASK,
 - kUSDHC_CommandTimeoutFlag = USDHC_INT_STATUS_CTOE_MASK,
 - kUSDHC_CommandCrcErrorFlag = USDHC_INT_STATUS_CCE_MASK,
 - kUSDHC_CommandEndBitErrorFlag = USDHC_INT_STATUS_CEBE_MASK,
 - kUSDHC_CommandIndexErrorFlag = USDHC_INT_STATUS_CIE_MASK,
 - kUSDHC_DataTimeoutFlag = USDHC_INT_STATUS_DTOE_MASK,
 - kUSDHC_DataCrcErrorFlag = USDHC_INT_STATUS_DCE_MASK,
 - kUSDHC_DataEndBitErrorFlag = USDHC_INT_STATUS_DEBE_MASK,
 - kUSDHC_AutoCommand12ErrorFlag = USDHC_INT_STATUS_AC12E_MASK,
 - kUSDHC_DmaErrorFlag = USDHC_INT_STATUS_DMAE_MASK,
 - kUSDHC_CommandErrorFlag,
 - kUSDHC_DataErrorFlag,
 - kUSDHC_ErrorFlag = (kUSDHC_CommandErrorFlag | kUSDHC_DataErrorFlag | kUSDHC_DmaErrorFlag),
 - kUSDHC_DataFlag,
 - kUSDHC_DataDMAFlag = (kUSDHC_DataCompleteFlag | kUSDHC_DataErrorFlag | kUSDHC_DmaErrorFlag),
 - kUSDHC_CommandFlag = (kUSDHC_CommandErrorFlag | kUSDHC_CommandCompleteFlag),
 - kUSDHC_CardDetectFlag = (kUSDHC_CardInsertionFlag | kUSDHC_CardRemovalFlag),
 - kUSDHC_SDR104TuningFlag = (kUSDHC_TuningErrorFlag | kUSDHC_TuningPassFlag | kUSDHC_ReTuningEventFlag),
 - kUSDHC_AllInterruptFlags }
 - Enum *_usdhc_interrupt_status_flag*.
 - enum {
 - kUSDHC_AutoCommand12NotExecutedFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12NE_MASK,
 - kUSDHC_AutoCommand12TimeoutFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12TOE_MASK,
 - kUSDHC_AutoCommand12EndBitErrorFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12EBE_MASK,
 - kUSDHC_AutoCommand12CrcErrorFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12CE_

```

MASK,
kUSDHC_AutoCommand12IndexErrorFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12IE-
_MASK,
kUSDHC_AutoCommand12NotIssuedFlag = USDHC_AUTOCMD12_ERR_STATUS_CNIBA-
C12E_MASK }
    Enum _usdhc_auto_command12_error_status_flag.
• enum {
kUSDHC_ExecuteTuning = USDHC_AUTOCMD12_ERR_STATUS_EXECUTE_TUNING_M-
ASK,
kUSDHC_TuningSampleClockSel }
    Enum _usdhc_standard_tuning.
• enum {
kUSDHC_AdmaLenghMismatchFlag = USDHC_ADMA_ERR_STATUS_ADMALME_MASK,
kUSDHC_AdmaDescriptorErrorFlag = USDHC_ADMA_ERR_STATUS_ADMADCE_MASK }
    Enum _usdhc_adma_error_status_flag.
• enum {
kUSDHC_AdmaErrorStateStopDma = 0x00U,
kUSDHC_AdmaErrorStateFetchDescriptor = 0x01U,
kUSDHC_AdmaErrorStateChangeAddress = 0x02U,
kUSDHC_AdmaErrorStateTransferData = 0x03U,
kUSDHC_AdmaErrorStateInvalidLength = 0x04U,
kUSDHC_AdmaErrorStateInvalidDescriptor = 0x08U,
kUSDHC_AdmaErrorState }
    Enum _usdhc_adma_error_state.
• enum {
kUSDHC_ForceEventAutoCommand12NotExecuted,
kUSDHC_ForceEventAutoCommand12Timeout = USDHC_FORCE_EVENT_FEVTAC12TOE_-
MASK,
kUSDHC_ForceEventAutoCommand12CrcError = USDHC_FORCE_EVENT_FEVTAC12CE_-
MASK,
kUSDHC_ForceEventEndBitError = USDHC_FORCE_EVENT_FEVTAC12EBE_MASK,
kUSDHC_ForceEventAutoCommand12IndexError = USDHC_FORCE_EVENT_FEVTAC12IE_-
MASK,
kUSDHC_ForceEventAutoCommand12NotIssued = USDHC_FORCE_EVENT_FEVTCNIBA-
C12E_MASK,
kUSDHC_ForceEventCommandTimeout = USDHC_FORCE_EVENT_FEVTC12TOE_MASK,
kUSDHC_ForceEventCommandCrcError = USDHC_FORCE_EVENT_FEVTC12CCE_MASK,
kUSDHC_ForceEventCommandEndBitError = USDHC_FORCE_EVENT_FEVTC12CEBE_MASK,
kUSDHC_ForceEventCommandIndexError = USDHC_FORCE_EVENT_FEVTC12CIE_MASK,
kUSDHC_ForceEventDataTimeout = USDHC_FORCE_EVENT_FEVTD12TOE_MASK,
kUSDHC_ForceEventDataCrcError = USDHC_FORCE_EVENT_FEVTD12CCE_MASK,
kUSDHC_ForceEventDataEndBitError = USDHC_FORCE_EVENT_FEVTD12EBE_MASK,
kUSDHC_ForceEventAutoCommand12Error = USDHC_FORCE_EVENT_FEVTAC12E_MAS-

```

- K,
- kUSDHC_ForceEventCardInt = (int)USDHC_FORCE_EVENT_FEVTCINT_MASK,
 - kUSDHC_ForceEventDmaError = USDHC_FORCE_EVENT_FEVTDMAE_MASK,
 - kUSDHC_ForceEventTuningError = USDHC_FORCE_EVENT_FEVTTNE_MASK,
 - kUSDHC_ForceEventsAll }
- Enum _usdhc_force_event.*
- enum usdhc_transfer_direction_t {
 - kUSDHC_TransferDirectionReceive = 1U,
 - kUSDHC_TransferDirectionSend = 0U }

Data transfer direction.
 - enum usdhc_data_bus_width_t {
 - kUSDHC_DataBusWidth1Bit = 0U,
 - kUSDHC_DataBusWidth4Bit = 1U,
 - kUSDHC_DataBusWidth8Bit = 2U }

Data transfer width.
 - enum usdhc_endian_mode_t {
 - kUSDHC_EndianModeBig = 0U,
 - kUSDHC_EndianModeHalfWordBig = 1U,
 - kUSDHC_EndianModeLittle = 2U }

Endian mode.
 - enum usdhc_dma_mode_t {
 - kUSDHC_DmaModeSimple = 0U,
 - kUSDHC_DmaModeAdma1 = 1U,
 - kUSDHC_DmaModeAdma2 = 2U,
 - kUSDHC_ExternalDMA = 3U }

DMA mode.
 - enum {
 - kUSDHC_StopAtBlockGapFlag = USDHC_PROT_CTRL_SABGREQ_MASK,
 - kUSDHC_ReadWaitControlFlag = USDHC_PROT_CTRL_RWCTL_MASK,
 - kUSDHC_InterruptAtBlockGapFlag = USDHC_PROT_CTRL_IABG_MASK,
 - kUSDHC_ReadDoneNo8CLK = USDHC_PROT_CTRL_RD_DONE_NO_8CLK_MASK,
 - kUSDHC_ExactBlockNumberReadFlag = USDHC_PROT_CTRL_NON_EXACT_BLK_RD_M-
ASK }

Enum _usdhc_sdio_control_flag.
 - enum usdhc_boot_mode_t {
 - kUSDHC_BootModeNormal = 0U,
 - kUSDHC_BootModeAlternative = 1U }

MMC card boot mode.
 - enum usdhc_card_command_type_t {
 - kCARD_CommandTypeNormal = 0U,
 - kCARD_CommandTypeSuspend = 1U,
 - kCARD_CommandTypeResume = 2U,
 - kCARD_CommandTypeAbort = 3U,
 - kCARD_CommandTypeEmpty = 4U }

The command type.
 - enum usdhc_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 {


```
kUSDHC_Adma1DescriptorValidFlag = (1U << 0U),
kUSDHC_Adma1DescriptorEndFlag = (1U << 1U),
kUSDHC_Adma1DescriptorInterruptFlag = (1U << 2U),
kUSDHC_Adma1DescriptorActivity1Flag = (1U << 4U),
kUSDHC_Adma1DescriptorActivity2Flag = (1U << 5U),
kUSDHC_Adma1DescriptorTypeNop = (kUSDHC_Adma1DescriptorValidFlag),
kUSDHC_Adma1DescriptorTypeTransfer = (kUSDHC_Adma1DescriptorActivity2Flag | kUSDH-
C_Adma1DescriptorValidFlag),
kUSDHC_Adma1DescriptorTypeLink,
kUSDHC_Adma1DescriptorTypeSetLength = (kUSDHC_Adma1DescriptorActivity1Flag | kUSD-
HC_Adma1DescriptorValidFlag) }
```

Enum _usdhc_adma1_descriptor_flag.

- enum {


```
kUSDHC_Adma2DescriptorValidFlag = (1U << 0U),
kUSDHC_Adma2DescriptorEndFlag = (1U << 1U),
kUSDHC_Adma2DescriptorInterruptFlag = (1U << 2U),
kUSDHC_Adma2DescriptorActivity1Flag = (1U << 4U),
kUSDHC_Adma2DescriptorActivity2Flag = (1U << 5U),
kUSDHC_Adma2DescriptorTypeNop = (kUSDHC_Adma2DescriptorValidFlag),
kUSDHC_Adma2DescriptorTypeReserved = (kUSDHC_Adma2DescriptorActivity1Flag | kUSD-
HC_Adma2DescriptorValidFlag),
kUSDHC_Adma2DescriptorTypeTransfer = (kUSDHC_Adma2DescriptorActivity2Flag | kUSDH-
C_Adma2DescriptorValidFlag),
kUSDHC_Adma2DescriptorTypeLink }
```

Enum _usdhc_adma2_descriptor_flag.

- enum {


```
kUSDHC_AdmaDescriptorSingleFlag = 0U,
kUSDHC_AdmaDescriptorMultipleFlag }
```

Enum _usdhc_adma_flag.

- enum usdhc_burst_len_t {


```
kUSDHC_EnBurstLenForINCR = 0x01U,
kUSDHC_EnBurstLenForINCR4816 = 0x02U,
kUSDHC_EnBurstLenForINCR4816WRAP = 0x04U }
```

DMA transfer burst len config.

- enum {
 kUSDHC_TransferDataNormal = 0U,
 kUSDHC_TransferDataTuning = 1U,
 kUSDHC_TransferDataBoot = 2U,
 kUSDHC_TransferDataBootcontinuous = 3U }
 Enum_usdhc_transfer_data_type.

Driver version

- #define FSL_USDHC_DRIVER_VERSION (MAKE_VERSION(2U, 8U, 2U))
 Driver version 2.8.2.

Initialization and deinitialization

- void USDHC_Init (USDHC_Type *base, const usdhc_config_t *config)
 USDHC module initialization function.
- void USDHC_Deinit (USDHC_Type *base)
 Deinitializes the USDHC.
- bool USDHC_Reset (USDHC_Type *base, uint32_t mask, uint32_t timeout)
 Resets the USDHC.

DMA Control

- status_t USDHC_SetAdmaTableConfig (USDHC_Type *base, usdhc_adma_config_t *dmaConfig, usdhc_data_t *dataConfig, uint32_t flags)
 Sets the DMA descriptor table configuration.
- status_t USDHC_SetInternalDmaConfig (USDHC_Type *base, usdhc_adma_config_t *dmaConfig, const uint32_t *dataAddr, bool enAutoCmd23)
 Internal DMA configuration.
- status_t USDHC_SetADMA2Descriptor (uint32_t *admaTable, uint32_t admaTableWords, const uint32_t *dataBufferAddr, uint32_t dataBytes, uint32_t flags)
 Sets the ADMA2 descriptor table configuration.
- status_t USDHC_SetADMA1Descriptor (uint32_t *admaTable, uint32_t admaTableWords, const uint32_t *dataBufferAddr, uint32_t dataBytes, uint32_t flags)
 Sets the ADMA1 descriptor table configuration.
- static void USDHC_EnableInternalDMA (USDHC_Type *base, bool enable)
 Enables internal DMA.

Interrupts

- static void USDHC_EnableInterruptStatus (USDHC_Type *base, uint32_t mask)
 Enables the interrupt status.
- static void USDHC_DisableInterruptStatus (USDHC_Type *base, uint32_t mask)
 Disables the interrupt status.
- static void USDHC_EnableInterruptSignal (USDHC_Type *base, uint32_t mask)
 Enables the interrupt signal corresponding to the interrupt status flag.
- static void USDHC_DisableInterruptSignal (USDHC_Type *base, uint32_t mask)
 Disables the interrupt signal corresponding to the interrupt status flag.

Status

- static uint32_t [USDHC_GetEnabledInterruptStatusFlags](#) (USDHC_Type *base)
Gets the enabled interrupt status.
- static uint32_t [USDHC_GetInterruptStatusFlags](#) (USDHC_Type *base)
Gets the current interrupt status.
- static void [USDHC_ClearInterruptStatusFlags](#) (USDHC_Type *base, uint32_t mask)
Clears a specified interrupt status.
- static uint32_t [USDHC_GetAutoCommand12ErrorStatusFlags](#) (USDHC_Type *base)
Gets the status of auto command 12 error.
- static uint32_t [USDHC_GetAdmaErrorStatusFlags](#) (USDHC_Type *base)
Gets the status of the ADMA error.
- static uint32_t [USDHC_GetPresentStatusFlags](#) (USDHC_Type *base)
Gets a present status.

Bus Operations

- void [USDHC_GetCapability](#) (USDHC_Type *base, [usdhc_capability_t](#) *capability)
Gets the capability information.
- static void [USDHC_ForceClockOn](#) (USDHC_Type *base, bool enable)
Forces the card clock on.
- uint32_t [USDHC_SetSdClock](#) (USDHC_Type *base, uint32_t srcClock_Hz, uint32_t busClock_Hz)
Sets the SD bus clock frequency.
- bool [USDHC_SetCardActive](#) (USDHC_Type *base, uint32_t timeout)
Sends 80 clocks to the card to set it to the active state.
- static void [USDHC_AssertHardwareReset](#) (USDHC_Type *base, bool high)
Triggers a hardware reset.
- static void [USDHC_SetDataBusWidth](#) (USDHC_Type *base, [usdhc_data_bus_width_t](#) width)
Sets the data transfer width.
- static void [USDHC_WriteData](#) (USDHC_Type *base, uint32_t data)
Fills the data port.
- static uint32_t [USDHC_ReadData](#) (USDHC_Type *base)
Retrieves the data from the data port.
- void [USDHC_SendCommand](#) (USDHC_Type *base, [usdhc_command_t](#) *command)
Sends command function.
- static void [USDHC_EnableWakeupEvent](#) (USDHC_Type *base, uint32_t mask, bool enable)
Enables or disables a wakeup event in low-power mode.
- static void [USDHC_CardDetectByData3](#) (USDHC_Type *base, bool enable)
Detects card insert status.
- static bool [USDHC_DetectCardInsert](#) (USDHC_Type *base)
Detects card insert status.
- static void [USDHC_EnableSdioControl](#) (USDHC_Type *base, uint32_t mask, bool enable)
Enables or disables the SDIO card control.
- static void [USDHC_SetContinueRequest](#) (USDHC_Type *base)
Restarts a transaction which has stopped at the block GAP for the SDIO card.
- static void [USDHC_RequestStopAtBlockGap](#) (USDHC_Type *base, bool enable)
Request stop at block gap function.
- void [USDHC_SetMmcBootConfig](#) (USDHC_Type *base, const [usdhc_boot_config_t](#) *config)
Configures the MMC boot feature.
- static void [USDHC_EnableMmcBoot](#) (USDHC_Type *base, bool enable)
Enables or disables the mmc boot mode.

- static void [USDHC_SetForceEvent](#) (USDHC_Type *base, uint32_t mask)
Forces generating events according to the given mask.
- static void [USDHC_SelectVoltage](#) (USDHC_Type *base, bool en18v)
Selects the USDHC output voltage.
- static bool [USDHC_RequestTuningForSDR50](#) (USDHC_Type *base)
Checks the SDR50 mode request tuning bit.
- static bool [USDHC_RequestReTuning](#) (USDHC_Type *base)
Checks the request re-tuning bit.
- static void [USDHC_EnableAutoTuning](#) (USDHC_Type *base, bool enable)
The SDR104 mode auto tuning enable and disable.
- void [USDHC_EnableAutoTuningForCmdAndData](#) (USDHC_Type *base)
The auto tuning enable for CMD/DATA line.
- void [USDHC_EnableManualTuning](#) (USDHC_Type *base, bool enable)
Manual tuning trigger or abort.
- static uint32_t [USDHC_GetTuningDelayStatus](#) (USDHC_Type *base)
Get the tuning delay cell setting.
- [status_t USDHC_SetTuningDelay](#) (USDHC_Type *base, uint32_t preDelay, uint32_t outDelay, uint32_t postDelay)
The tuning delay cell setting.
- [status_t USDHC_AdjustDelayForManualTuning](#) (USDHC_Type *base, uint32_t delay)
Adjusts delay for manual tuning.
- static void [USDHC_SetStandardTuningCounter](#) (USDHC_Type *base, uint8_t counter)
set tuning counter tuning.
- void [USDHC_EnableStandardTuning](#) (USDHC_Type *base, uint32_t tuningStartTap, uint32_t step, bool enable)
The enable standard tuning function.
- static uint32_t [USDHC_GetExecuteStdTuningStatus](#) (USDHC_Type *base)
Gets execute STD tuning status.
- static uint32_t [USDHC_CheckStdTuningResult](#) (USDHC_Type *base)
Checks STD tuning result.
- static uint32_t [USDHC_CheckTuningError](#) (USDHC_Type *base)
Checks tuning error.
- void [USDHC_EnableDDRMMode](#) (USDHC_Type *base, bool enable, uint32_t nibblePos)
The enable/disable DDR mode.
- static void [USDHC_EnableHS400Mode](#) (USDHC_Type *base, bool enable)
The enable/disable HS400 mode.
- static void [USDHC_ResetStrobeDLL](#) (USDHC_Type *base)
Resets the strobe DLL.
- static void [USDHC_EnableStrobeDLL](#) (USDHC_Type *base, bool enable)
Enables/disables the strobe DLL.
- void [USDHC_ConfigStrobeDLL](#) (USDHC_Type *base, uint32_t delayTarget, uint32_t updateInterval)
Configs the strobe DLL delay target and update interval.
- static void [USDHC_SetStrobeDllOverride](#) (USDHC_Type *base, uint32_t delayTaps)
*Enables manual override for slave delay chain using **STROBE_SLV_OVERRIDE_VAL**.*
- static uint32_t [USDHC_GetStrobeDLLStatus](#) (USDHC_Type *base)
Gets the strobe DLL status.
- void [USDHC_SetDataConfig](#) (USDHC_Type *base, [usdhc_transfer_direction_t](#) dataDirection, uint32_t blockCount, uint32_t blockSize)
USDHC data configuration.

Transactional functions

- void [USDHC_TransferCreateHandle](#) (USDHC_Type *base, usdhc_handle_t *handle, const [usdhc_transfer_callback_t](#) *callback, void *userData)
Creates the USDHC handle.
- [status_t USDHC_TransferNonBlocking](#) (USDHC_Type *base, usdhc_handle_t *handle, [usdhc_adma_config_t](#) *dmaConfig, [usdhc_transfer_t](#) *transfer)
Transfers the command/data using an interrupt and an asynchronous method.
- [status_t USDHC_TransferBlocking](#) (USDHC_Type *base, [usdhc_adma_config_t](#) *dmaConfig, [usdhc_transfer_t](#) *transfer)
Transfers the command/data using a blocking method.
- void [USDHC_TransferHandleIRQ](#) (USDHC_Type *base, usdhc_handle_t *handle)
IRQ handler for the USDHC.

46.3 Data Structure Documentation

46.3.1 struct usdhc_adma2_descriptor_t

Data Fields

- [uint32_t attribute](#)
The control and status field.
- const [uint32_t](#) * [address](#)
The address field.

Field Documentation

(1) [uint32_t usdhc_adma2_descriptor_t::attribute](#)

(2) [const uint32_t* usdhc_adma2_descriptor_t::address](#)

46.3.2 struct usdhc_capability_t

Defines a structure to save the capability information of USDHC.

Data Fields

- [uint32_t sdVersion](#)
Support SD card/sdio version.
- [uint32_t mmcVersion](#)
Support EMMC card 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([_usdhc_capability_flag](#)).

Field Documentation

- (1) `uint32_t usdhc_capability_t::sdVersion`
- (2) `uint32_t usdhc_capability_t::mmcVersion`
- (3) `uint32_t usdhc_capability_t::maxBlockLength`
- (4) `uint32_t usdhc_capability_t::maxBlockCount`
- (5) `uint32_t usdhc_capability_t::flags`

46.3.3 struct usdhc_boot_config_t**Data Fields**

- `uint32_t ackTimeoutCount`
Timeout value for the boot ACK.
- `usdhc_boot_mode_t bootMode`
Boot mode selection.
- `uint32_t blockCount`
Stop at block gap value of automatic mode.
- `size_t blockSize`
Block size.
- `bool enableBootAck`
Enable or disable boot ACK.
- `bool enableAutoStopAtBlockGap`
Enable or disable auto stop at block gap function in boot period.

Field Documentation

- (1) `uint32_t usdhc_boot_config_t::ackTimeoutCount`

The available range is 0 ~ 15.

- (2) `usdhc_boot_mode_t usdhc_boot_config_t::bootMode`

- (3) `uint32_t usdhc_boot_config_t::blockCount`

Available range is 0 ~ 65535.

- (4) `size_t usdhc_boot_config_t::blockSize`
- (5) `bool usdhc_boot_config_t::enableBootAck`
- (6) `bool usdhc_boot_config_t::enableAutoStopAtBlockGap`

46.3.4 struct `usdhc_config_t`

Data Fields

- `uint32_t dataTimeout`
Data timeout value.
- `usdhc_endian_mode_t endianMode`
Endian mode.
- `uint8_t readWatermarkLevel`
Watermark level for DMA read operation.
- `uint8_t writeWatermarkLevel`
Watermark level for DMA write operation.

Field Documentation

- (1) `uint32_t usdhc_config_t::dataTimeout`
- (2) `usdhc_endian_mode_t usdhc_config_t::endianMode`
- (3) `uint8_t usdhc_config_t::readWatermarkLevel`

Available range is 1 ~ 128.

- (4) `uint8_t usdhc_config_t::writeWatermarkLevel`

Available range is 1 ~ 128.

46.3.5 struct `usdhc_command_t`

Defines card command-related attribute.

Data Fields

- `uint32_t index`
Command index.
- `uint32_t argument`
Command argument.
- `usdhc_card_command_type_t type`
Command type.
- `usdhc_card_response_type_t responseType`
Command response type.

- `uint32_t response` [4U]
Response for this command.
- `uint32_t responseErrorFlags`
Response error flag, which need to check the command reponse.
- `uint32_t flags`
Cmd flags.

Field Documentation

- (1) `uint32_t usdhc_command_t::index`
- (2) `uint32_t usdhc_command_t::argument`
- (3) `usdhc_card_command_type_t usdhc_command_t::type`
- (4) `usdhc_card_response_type_t usdhc_command_t::responseType`
- (5) `uint32_t usdhc_command_t::response[4U]`
- (6) `uint32_t usdhc_command_t::responseErrorFlags`
- (7) `uint32_t usdhc_command_t::flags`

46.3.6 struct `usdhc_adma_config_t`

Data Fields

- `usdhc_dma_mode_t dmaMode`
DMA mode.
- `uint32_t * admaTable`
ADMA table address, can't be null if transfer way is ADMA1/ADMA2.
- `uint32_t admaTableWords`
ADMA table length united as words, can't be 0 if transfer way is ADMA1/ADMA2.

Field Documentation

- (1) `usdhc_dma_mode_t usdhc_adma_config_t::dmaMode`
- (2) `uint32_t* usdhc_adma_config_t::admaTable`
- (3) `uint32_t usdhc_adma_config_t::admaTableWords`

46.3.7 struct `usdhc_scatter_gather_data_list_t`

Allow application register uncontinuous data buffer for data transfer.

46.3.8 struct usdhc_scatter_gather_data_t

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

Data Fields

- bool [enableAutoCommand12](#)
Enable auto CMD12.
- bool [enableAutoCommand23](#)
Enable auto CMD23.
- bool [enableIgnoreError](#)
Enable to ignore error event to read/write all the data.
- [usdhc_transfer_direction_t](#) [dataDirection](#)
data direction
- [uint8_t](#) [dataType](#)
this is used to distinguish the normal/tuning/boot data.
- [size_t](#) [blockSize](#)
Block size.
- [usdhc_scatter_gather_data_list_t](#) [sgData](#)
scatter gather data

Field Documentation

- (1) **bool usdhc_scatter_gather_data_t::enableAutoCommand12**
- (2) **bool usdhc_scatter_gather_data_t::enableAutoCommand23**
- (3) **bool usdhc_scatter_gather_data_t::enableIgnoreError**
- (4) **uint8_t usdhc_scatter_gather_data_t::dataType**
- (5) **size_t usdhc_scatter_gather_data_t::blockSize**

46.3.9 struct usdhc_scatter_gather_transfer_t

Data Fields

- [usdhc_scatter_gather_data_t](#) * [data](#)
Data to transfer.
- [usdhc_command_t](#) * [command](#)
Command to send.

Field Documentation

(1) `usdhc_scatter_gather_data_t* usdhc_scatter_gather_transfer_t::data`

(2) `usdhc_command_t* usdhc_scatter_gather_transfer_t::command`

46.3.10 struct `usdhc_data_t`

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

Data Fields

- bool `enableAutoCommand12`
Enable auto CMD12.
- bool `enableAutoCommand23`
Enable auto CMD23.
- bool `enableIgnoreError`
Enable to ignore error event to read/write all the data.
- uint8_t `dataType`
this is used to distinguish the normal/tuning/boot 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.

Field Documentation

- (1) `bool usdhc_data_t::enableAutoCommand12`
- (2) `bool usdhc_data_t::enableAutoCommand23`
- (3) `bool usdhc_data_t::enableIgnoreError`
- (4) `uint8_t usdhc_data_t::dataType`
- (5) `size_t usdhc_data_t::blockSize`
- (6) `uint32_t usdhc_data_t::blockCount`
- (7) `uint32_t* usdhc_data_t::rxData`
- (8) `const uint32_t* usdhc_data_t::txData`

46.3.11 `struct usdhc_transfer_t`

Data Fields

- `usdhc_data_t * data`
Data to transfer.
- `usdhc_command_t * command`
Command to send.

Field Documentation

- (1) `usdhc_data_t* usdhc_transfer_t::data`
- (2) `usdhc_command_t* usdhc_transfer_t::command`

46.3.12 `struct usdhc_transfer_callback_t`

Data Fields

- `void(* CardInserted)(USDHC_Type *base, void *userData)`
Card inserted occurs when DAT3/CD pin is for card detect.
- `void(* CardRemoved)(USDHC_Type *base, void *userData)`
Card removed occurs.
- `void(* SdioInterrupt)(USDHC_Type *base, void *userData)`
SDIO card interrupt occurs.
- `void(* BlockGap)(USDHC_Type *base, void *userData)`
stopped at block gap event
- `void(* TransferComplete)(USDHC_Type *base, usdhc_handle_t *handle, status_t status, void *userData)`
Transfer complete callback.
- `void(* ReTuning)(USDHC_Type *base, void *userData)`

Handle the re-tuning.

Field Documentation

- (1) `void(* usdhc_transfer_callback_t::TransferComplete)(USDHC_Type *base, usdhc_handle_t *handle, status_t status, void *userData)`
- (2) `void(* usdhc_transfer_callback_t::ReTuning)(USDHC_Type *base, void *userData)`

46.3.13 struct _usdhc_handle

USDHC handle typedef.

Defines the structure to save the USDHC state information and callback function.

Note

All the fields except interruptFlags and transferredWords must be allocated by the user.

Data Fields

- `usdhc_data_t` *volatile data
Transfer parameter.
- `usdhc_command_t` *volatile command
Transfer parameter.
- volatile uint32_t transferredWords
Transfer status.
- `usdhc_transfer_callback_t` callback
Callback function.
- void * `userData`
Parameter for transfer complete callback.

Field Documentation

- (1) `usdhc_data_t`* volatile `usdhc_handle_t::data`

Data to transfer.

- (2) `usdhc_command_t`* volatile `usdhc_handle_t::command`

Command to send.

- (3) volatile uint32_t `usdhc_handle_t::transferredWords`

Words transferred by DATAPORT way.

(4) `usdhc_transfer_callback_t usdhc_handle_t::callback`

(5) `void* usdhc_handle_t::userData`

46.3.14 struct `usdhc_host_t`

Data Fields

- `USDHC_Type * base`
USDHC peripheral base address.
- `uint32_t sourceClock_Hz`
USDHC source clock frequency united in Hz.
- `usdhc_config_t config`
USDHC configuration.
- `usdhc_capability_t capability`
USDHC capability information.
- `usdhc_transfer_function_t transfer`
USDHC transfer function.

Field Documentation

(1) `USDHC_Type* usdhc_host_t::base`

(2) `uint32_t usdhc_host_t::sourceClock_Hz`

(3) `usdhc_config_t usdhc_host_t::config`

(4) `usdhc_capability_t usdhc_host_t::capability`

(5) `usdhc_transfer_function_t usdhc_host_t::transfer`

46.4 Macro Definition Documentation

46.4.1 `#define FSL_USDHC_DRIVER_VERSION (MAKE_VERSION(2U, 8U, 2U))`

46.4.2 `#define USDHC_ADMA1_ADDRESS_ALIGN (4096U)`

46.4.3 `#define USDHC_ADMA1_LENGTH_ALIGN (4096U)`

46.4.4 `#define USDHC_ADMA2_ADDRESS_ALIGN (4U)`

46.4.5 `#define USDHC_ADMA2_LENGTH_ALIGN (4U)`

46.4.6 `#define USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT (12U)`

Address/page field	Reserved	6p(*8-*3)*6/8lightgrayAttribute				
31 12	11 6	05	04	03	02	0
address or data length	000000	Act2	Act1	0	Int	0

ADMA1 descriptor table

Act2	Act1	Comment	31-28	27-12
0	0	No op	Don't care	
0	1	Set data length	0000	Data Length
1	0	Transfer data	Data address	
1	1	Link descriptor	Descriptor address	

ADMA2 action

46.4.7 #define USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK (0xFFFFFU)

46.4.8 #define USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT (12U)

46.4.9 #define USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK (0xFFFFU)

46.4.10 #define USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY (USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK + 1U - 4096U)

Since the max transfer size ADMA1 support is 65535 which is indivisible by 4096, so to make sure a large data load transfer (>64KB) continuously (require the data address be always align with 4096), software will set the maximum data length for ADMA1 to (64 - 4)KB.

46.4.11 #define USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT (16U)

Address field	Length	Reserved	6p(*9-*4)*6/9lightgrayAttribute			
63 32	31 16	15 06	05	04	03	02
32-bit address	16-bit length	0000000000	Act2	Act1	0	Int

ADMA2 descriptor table

46.4.12 #define USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK (0xFFFFU)

Act2	Act1	Comment	Operation
0	0	No op	Don't care
0	1	Reserved	Read this line and go to next one
1	0	Transfer data	Transfer data with address and length set in this descriptor line
1	1	Link descriptor	Link to another descriptor

ADMA2 action

46.6 Enumeration Type Documentation

46.6.1 anonymous enum

USDHC status.

Enumerator

- kStatus_USDHC_BusyTransferring* Transfer is on-going.
- kStatus_USDHC_PrepAdmaDescriptorFailed* Set DMA descriptor failed.
- kStatus_USDHC_SendCommandFailed* Send command failed.
- kStatus_USDHC_TransferDataFailed* Transfer data failed.
- kStatus_USDHC_DMADDataAddrNotAlign* Data address not aligned.
- kStatus_USDHC_ReTuningRequest* Re-tuning request.
- kStatus_USDHC_TuningError* Tuning error.
- kStatus_USDHC_NotSupport* Not support.
- kStatus_USDHC_TransferDataComplete* Transfer data complete.
- kStatus_USDHC_SendCommandSuccess* Transfer command complete.
- kStatus_USDHC_TransferDMAComplete* Transfer DMA complete.

46.6.2 anonymous enum

Host controller capabilities flag mask.

Enumerator

- kUSDHC_SupportAdmaFlag* Support ADMA.
- kUSDHC_SupportHighSpeedFlag* Support high-speed.
- kUSDHC_SupportDmaFlag* Support DMA.
- kUSDHC_SupportSuspendResumeFlag* Support suspend/resume.
- kUSDHC_SupportV330Flag* Support voltage 3.3V.

kUSDHC_SupportV300Flag Support voltage 3.0V.
kUSDHC_SupportV180Flag Support voltage 1.8V.
kUSDHC_Support4BitFlag Flag in HTCAPBLT_MBL's position, supporting 4-bit mode.
kUSDHC_Support8BitFlag Flag in HTCAPBLT_MBL's position, supporting 8-bit mode.
kUSDHC_SupportDDR50Flag SD version 3.0 new feature, supporting DDR50 mode.
kUSDHC_SupportSDR104Flag Support SDR104 mode.
kUSDHC_SupportSDR50Flag Support SDR50 mode.

46.6.3 anonymous enum

Wakeup event mask.

Enumerator

kUSDHC_WakeupEventOnCardInt Wakeup on card interrupt.
kUSDHC_WakeupEventOnCardInsert Wakeup on card insertion.
kUSDHC_WakeupEventOnCardRemove Wakeup on card removal.
kUSDHC_WakeupEventsAll All wakeup events.

46.6.4 anonymous enum

Reset type mask.

Enumerator

kUSDHC_ResetAll Reset all except card detection.
kUSDHC_ResetCommand Reset command line.
kUSDHC_ResetData Reset data line.
kUSDHC_ResetTuning Reset tuning circuit.
kUSDHC_ResetsAll All reset types.

46.6.5 anonymous enum

Transfer flag mask.

Enumerator

kUSDHC_EnableDmaFlag Enable DMA.
kUSDHC_CommandTypeSuspendFlag Suspend command.
kUSDHC_CommandTypeResumeFlag Resume command.
kUSDHC_CommandTypeAbortFlag Abort command.
kUSDHC_EnableBlockCountFlag Enable block count.
kUSDHC_EnableAutoCommand12Flag Enable auto CMD12.
kUSDHC_DataReadFlag Enable data read.

kUSDHC_MultipleBlockFlag Multiple block data read/write.
kUSDHC_EnableAutoCommand23Flag Enable auto CMD23.
kUSDHC_ResponseLength136Flag 136-bit response length.
kUSDHC_ResponseLength48Flag 48-bit response length.
kUSDHC_ResponseLength48BusyFlag 48-bit response length with busy status.
kUSDHC_EnableCrcCheckFlag Enable CRC check.
kUSDHC_EnableIndexCheckFlag Enable index check.
kUSDHC_DataPresentFlag Data present flag.

46.6.6 anonymous enum

Present status flag mask.

Enumerator

kUSDHC_CommandInhibitFlag Command inhibit.
kUSDHC_DataInhibitFlag Data inhibit.
kUSDHC_DataLineActiveFlag Data line active.
kUSDHC_SdClockStableFlag SD bus clock stable.
kUSDHC_WriteTransferActiveFlag Write transfer active.
kUSDHC_ReadTransferActiveFlag Read transfer active.
kUSDHC_BufferWriteEnableFlag Buffer write enable.
kUSDHC_BufferReadEnableFlag Buffer read enable.
kUSDHC_ReTuningRequestFlag Re-tuning request flag, only used for SDR104 mode.
kUSDHC_DelaySettingFinishedFlag Delay setting finished flag.
kUSDHC_CardInsertedFlag Card inserted.
kUSDHC_CommandLineLevelFlag Command line signal level.
kUSDHC_Data0LineLevelFlag Data0 line signal level.
kUSDHC_Data1LineLevelFlag Data1 line signal level.
kUSDHC_Data2LineLevelFlag Data2 line signal level.
kUSDHC_Data3LineLevelFlag Data3 line signal level.
kUSDHC_Data4LineLevelFlag Data4 line signal level.
kUSDHC_Data5LineLevelFlag Data5 line signal level.
kUSDHC_Data6LineLevelFlag Data6 line signal level.
kUSDHC_Data7LineLevelFlag Data7 line signal level.

46.6.7 anonymous enum

Interrupt status flag mask.

Enumerator

kUSDHC_CommandCompleteFlag Command complete.
kUSDHC_DataCompleteFlag Data complete.

kUSDHC_BlockGapEventFlag Block gap event.
kUSDHC_DmaCompleteFlag DMA interrupt.
kUSDHC_BufferWriteReadyFlag Buffer write ready.
kUSDHC_BufferReadReadyFlag Buffer read ready.
kUSDHC_CardInsertionFlag Card inserted.
kUSDHC_CardRemovalFlag Card removed.
kUSDHC_CardInterruptFlag Card interrupt.
kUSDHC_ReTuningEventFlag Re-Tuning event, only for SD3.0 SDR104 mode.
kUSDHC_TuningPassFlag SDR104 mode tuning pass flag.
kUSDHC_TuningErrorFlag SDR104 tuning error flag.
kUSDHC_CommandTimeoutFlag Command timeout error.
kUSDHC_CommandCrcErrorFlag Command CRC error.
kUSDHC_CommandEndBitErrorFlag Command end bit error.
kUSDHC_CommandIndexErrorFlag Command index error.
kUSDHC_DataTimeoutFlag Data timeout error.
kUSDHC_DataCrcErrorFlag Data CRC error.
kUSDHC_DataEndBitErrorFlag Data end bit error.
kUSDHC_AutoCommand12ErrorFlag Auto CMD12 error.
kUSDHC_DmaErrorFlag DMA error.
kUSDHC_CommandErrorFlag Command error.
kUSDHC_DataErrorFlag Data error.
kUSDHC_ErrorFlag All error.
kUSDHC_DataFlag Data interrupts.
kUSDHC_DataDMAFlag Data interrupts.
kUSDHC_CommandFlag Command interrupts.
kUSDHC_CardDetectFlag Card detection interrupts.
kUSDHC_SDR104TuningFlag SDR104 tuning flag.
kUSDHC_AllInterruptFlags All flags mask.

46.6.8 anonymous enum

Auto CMD12 error status flag mask.

Enumerator

kUSDHC_AutoCommand12NotExecutedFlag Not executed error.
kUSDHC_AutoCommand12TimeoutFlag Timeout error.
kUSDHC_AutoCommand12EndBitErrorFlag End bit error.
kUSDHC_AutoCommand12CrcErrorFlag CRC error.
kUSDHC_AutoCommand12IndexErrorFlag Index error.
kUSDHC_AutoCommand12NotIssuedFlag Not issued error.

46.6.9 anonymous enum

Standard tuning flag.

Enumerator

kUSDHC_ExecuteTuning Used to start tuning procedure.

kUSDHC_TuningSampleClockSel When `std_tuning_en` bit is set, this bit is used to select sampling clock.

46.6.10 anonymous enum

ADMA error status flag mask.

Enumerator

kUSDHC_AdmaLenghMismatchFlag Length mismatch error.

kUSDHC_AdmaDescriptorErrorFlag Descriptor error.

46.6.11 anonymous enum

ADMA error state.

This state is the detail state when ADMA error has occurred.

Enumerator

kUSDHC_AdmaErrorStateStopDma Stop DMA, previous location set in the ADMA system address is errored address.

kUSDHC_AdmaErrorStateFetchDescriptor Fetch descriptor, current location set in the ADMA system address is errored address.

kUSDHC_AdmaErrorStateChangeAddress Change address, no DMA error has occurred.

kUSDHC_AdmaErrorStateTransferData Transfer data, previous location set in the ADMA system address is errored address.

kUSDHC_AdmaErrorStateInvalidLength Invalid length in ADMA descriptor.

kUSDHC_AdmaErrorStateInvalidDescriptor Invalid descriptor fetched by ADMA.

kUSDHC_AdmaErrorState ADMA error state.

46.6.12 anonymous enum

Force event bit position.

Enumerator

kUSDHC_ForceEventAutoCommand12NotExecuted Auto CMD12 not executed error.

kUSDHC_ForceEventAutoCommand12Timeout Auto CMD12 timeout error.
kUSDHC_ForceEventAutoCommand12CrcError Auto CMD12 CRC error.
kUSDHC_ForceEventEndBitError Auto CMD12 end bit error.
kUSDHC_ForceEventAutoCommand12IndexError Auto CMD12 index error.
kUSDHC_ForceEventAutoCommand12NotIssued Auto CMD12 not issued error.
kUSDHC_ForceEventCommandTimeout Command timeout error.
kUSDHC_ForceEventCommandCrcError Command CRC error.
kUSDHC_ForceEventCommandEndBitError Command end bit error.
kUSDHC_ForceEventCommandIndexError Command index error.
kUSDHC_ForceEventDataTimeout Data timeout error.
kUSDHC_ForceEventDataCrcError Data CRC error.
kUSDHC_ForceEventDataEndBitError Data end bit error.
kUSDHC_ForceEventAutoCommand12Error Auto CMD12 error.
kUSDHC_ForceEventCardInt Card interrupt.
kUSDHC_ForceEventDmaError Dma error.
kUSDHC_ForceEventTuningError Tuning error.
kUSDHC_ForceEventsAll All force event flags mask.

46.6.13 enum usdhc_transfer_direction_t

Enumerator

kUSDHC_TransferDirectionReceive USDHC transfer direction receive.
kUSDHC_TransferDirectionSend USDHC transfer direction send.

46.6.14 enum usdhc_data_bus_width_t

Enumerator

kUSDHC_DataBusWidth1Bit 1-bit mode
kUSDHC_DataBusWidth4Bit 4-bit mode
kUSDHC_DataBusWidth8Bit 8-bit mode

46.6.15 enum usdhc_endian_mode_t

Enumerator

kUSDHC_EndianModeBig Big endian mode.
kUSDHC_EndianModeHalfWordBig Half word big endian mode.
kUSDHC_EndianModeLittle Little endian mode.

46.6.16 enum usdhc_dma_mode_t

Enumerator

- kUSDHC_DmaModeSimple* External DMA.
- kUSDHC_DmaModeAdma1* ADMA1 is selected.
- kUSDHC_DmaModeAdma2* ADMA2 is selected.
- kUSDHC_ExternalDMA* External DMA mode selected.

46.6.17 anonymous enum

SDIO control flag mask.

Enumerator

- kUSDHC_StopAtBlockGapFlag* Stop at block gap.
- kUSDHC_ReadWaitControlFlag* Read wait control.
- kUSDHC_InterruptAtBlockGapFlag* Interrupt at block gap.
- kUSDHC_ReadDoneNo8CLK* Read done without 8 clk for block gap.
- kUSDHC_ExactBlockNumberReadFlag* Exact block number read.

46.6.18 enum usdhc_boot_mode_t

Enumerator

- kUSDHC_BootModeNormal* Normal boot.
- kUSDHC_BootModeAlternative* Alternative boot.

46.6.19 enum usdhc_card_command_type_t

Enumerator

- kCARD_CommandTypeNormal* Normal command.
- kCARD_CommandTypeSuspend* Suspend command.
- kCARD_CommandTypeResume* Resume command.
- kCARD_CommandTypeAbort* Abort command.
- kCARD_CommandTypeEmpty* Empty command.

46.6.20 enum usdhc_card_response_type_t

Defines 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.

46.6.21 anonymous enum

The mask for the control/status field in ADMA1 descriptor.

Enumerator

kUSDHC_Adma1DescriptorValidFlag Valid flag.
kUSDHC_Adma1DescriptorEndFlag End flag.
kUSDHC_Adma1DescriptorInterruptFlag Interrupt flag.
kUSDHC_Adma1DescriptorActivity1Flag Activity 1 flag.
kUSDHC_Adma1DescriptorActivity2Flag Activity 2 flag.
kUSDHC_Adma1DescriptorTypeNop No operation.
kUSDHC_Adma1DescriptorTypeTransfer Transfer data.
kUSDHC_Adma1DescriptorTypeLink Link descriptor.
kUSDHC_Adma1DescriptorTypeSetLength Set data length.

46.6.22 anonymous enum

ADMA1 descriptor control and status mask.

Enumerator

kUSDHC_Adma2DescriptorValidFlag Valid flag.
kUSDHC_Adma2DescriptorEndFlag End flag.
kUSDHC_Adma2DescriptorInterruptFlag Interrupt flag.
kUSDHC_Adma2DescriptorActivity1Flag Activity 1 mask.
kUSDHC_Adma2DescriptorActivity2Flag Activity 2 mask.
kUSDHC_Adma2DescriptorTypeNop No operation.
kUSDHC_Adma2DescriptorTypeReserved Reserved.
kUSDHC_Adma2DescriptorTypeTransfer Transfer type.
kUSDHC_Adma2DescriptorTypeLink Link type.

46.6.23 anonymous enum

ADMA descriptor configuration flag.

Enumerator

kUSDHC_AdmaDescriptorSingleFlag Try to finish the transfer in a single ADMA descriptor. If transfer size is bigger than one ADMA descriptor's ability, new another descriptor for data transfer.

kUSDHC_AdmaDescriptorMultipleFlag Create multiple ADMA descriptors within the ADMA table, this is used for mmc boot mode specifically, which need to modify the ADMA descriptor on the fly, so the flag should be used combining with stop at block gap feature.

46.6.24 enum usdhc_burst_len_t

Enumerator

kUSDHC_EnBurstLenForINCR Enable burst len for INCR.

kUSDHC_EnBurstLenForINCR4816 Enable burst len for INCR4/INCR8/INCR16.

kUSDHC_EnBurstLenForINCR4816WRAP Enable burst len for INCR4/8/16 WRAP.

46.6.25 anonymous enum

Transfer data type definition.

Enumerator

kUSDHC_TransferDataNormal Transfer normal read/write data.

kUSDHC_TransferDataTuning Transfer tuning data.

kUSDHC_TransferDataBoot Transfer boot data.

kUSDHC_TransferDataBootcontinuous Transfer boot data continuously.

46.7 Function Documentation

46.7.1 void USDHC_Init (USDHC_Type * base, const usdhc_config_t * config)

Configures the USDHC according to the user configuration.

Example:

```
usdhc_config_t config;
config.cardDetectDat3 = false;
config.endianMode = kUSDHC_EndianModeLittle;
config.dmaMode = kUSDHC_DmaModeAdma2;
config.readWatermarkLevel = 128U;
config.writeWatermarkLevel = 128U;
USDHC_Init(USDHC, &config);
```

Parameters

<i>base</i>	USDHC peripheral base address.
<i>config</i>	USDHC configuration information.

Return values

<i>kStatus_Success</i>	Operate successfully.
------------------------	-----------------------

46.7.2 void USDHC_Deinit (USDHC_Type * *base*)

Parameters

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

46.7.3 bool USDHC_Reset (USDHC_Type * *base*, uint32_t *mask*, uint32_t *timeout*)

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The reset type mask(_usdhc_reset).
<i>timeout</i>	Timeout for reset.

Return values

<i>true</i>	Reset successfully.
<i>false</i>	Reset failed.

46.7.4 status_t USDHC_SetAdmaTableConfig (USDHC_Type * *base*, usdhc_adma_config_t * *dmaConfig*, usdhc_data_t * *dataConfig*, uint32_t *flags*)

A high level DMA descriptor configuration function.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>dmaConfig</i>	ADMA configuration
<i>dataConfig</i>	Data descriptor
<i>flags</i>	ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum _usdhc_adma_flag .

Return values

kStatus_OutOfRange	ADMA descriptor table length isn't enough to describe data.
kStatus_Success	Operate successfully.

46.7.5 status_t USDHC_SetInternalDmaConfig (USDHC_Type * *base*, usdhc_adma_config_t * *dmaConfig*, const uint32_t * *dataAddr*, bool *enAutoCmd23*)

This function is used to config the USDHC DMA related registers.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>dmaConfig</i>	ADMA configuration.
<i>dataAddr</i>	Transfer data address, a simple DMA parameter, if ADMA is used, leave it to NULL.
<i>enAutoCmd23</i>	Flag to indicate Auto CMD23 is enable or not, a simple DMA parameter, if ADMA is used, leave it to false.

Return values

kStatus_OutOfRange	ADMA descriptor table length isn't enough to describe data.
kStatus_Success	Operate successfully.

46.7.6 status_t USDHC_SetADMA2Descriptor (uint32_t * *admaTable*, uint32_t *admaTableWords*, const uint32_t * *dataBufferAddr*, uint32_t *dataBytes*, uint32_t *flags*)

Parameters

<i>admaTable</i>	ADMA table address.
<i>admaTable- Words</i>	ADMA table length.
<i>dataBufferAddr</i>	Data buffer address.
<i>dataBytes</i>	Data Data length.
<i>flags</i>	ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum _usdhc_adma_flag .

Return values

<i>kStatus_OutOfRange</i>	ADMA descriptor table length isn't enough to describe data.
<i>kStatus_Success</i>	Operate successfully.

46.7.7 `status_t USDHC_SetADMA1Descriptor (uint32_t * admaTable, uint32_t admaTableWords, const uint32_t * dataBufferAddr, uint32_t dataBytes, uint32_t flags)`

Parameters

<i>admaTable</i>	ADMA table address.
<i>admaTable- Words</i>	ADMA table length.
<i>dataBufferAddr</i>	Data buffer address.
<i>dataBytes</i>	Data length.
<i>flags</i>	ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum _usdhc_adma_flag .

Return values

<i>kStatus_OutOfRange</i>	ADMA descriptor table length isn't enough to describe data.
<i>kStatus_Success</i>	Operate successfully.

46.7.8 `static void USDHC_EnableInternalDMA (USDHC_Type * base, bool enable) [inline], [static]`

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	enable or disable flag

46.7.9 static void USDHC_EnableInterruptStatus (USDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	Interrupt status flags mask(_usdhc_interrupt_status_flag).

46.7.10 static void USDHC_DisableInterruptStatus (USDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The interrupt status flags mask(_usdhc_interrupt_status_flag).

46.7.11 static void USDHC_EnableInterruptSignal (USDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The interrupt status flags mask(_usdhc_interrupt_status_flag).

46.7.12 static void USDHC_DisableInterruptSignal (USDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The interrupt status flags mask(_usdhc_interrupt_status_flag).

46.7.13 `static uint32_t USDHC_GetEnabledInterruptStatusFlags (USDHC_Type * base) [inline], [static]`

Parameters

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

Returns

Current interrupt status flags mask([_usdhc_interrupt_status_flag](#)).

46.7.14 `static uint32_t USDHC_GetInterruptStatusFlags (USDHC_Type * base) [inline], [static]`

Parameters

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

Returns

Current interrupt status flags mask([_usdhc_interrupt_status_flag](#)).

46.7.15 `static void USDHC_ClearInterruptStatusFlags (USDHC_Type * base, uint32_t mask) [inline], [static]`

write 1 clears

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The interrupt status flags mask(_usdhc_interrupt_status_flag).

46.7.16 **static uint32_t USDHC_GetAutoCommand12ErrorStatusFlags (USDHC_Type * *base*) [inline], [static]**

Parameters

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

Returns

Auto command 12 error status flags mask([_usdhc_auto_command12_error_status_flag](#)).

46.7.17 **static uint32_t USDHC_GetAdmaErrorStatusFlags (USDHC_Type * *base*) [inline], [static]**

Parameters

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

Returns

ADMA error status flags mask([_usdhc_adma_error_status_flag](#)).

46.7.18 **static uint32_t USDHC_GetPresentStatusFlags (USDHC_Type * *base*) [inline], [static]**

This function gets the present USDHC's status except for an interrupt status and an error status.

Parameters

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

Returns

Present USDHC's status flags mask([_usdhc_present_status_flag](#)).

46.7.19 void USDHC_GetCapability (USDHC_Type * *base*, usdhc_capability_t * *capability*)

Parameters

<i>base</i>	USDHC peripheral base address.
<i>capability</i>	Structure to save capability information.

**46.7.20 static void USDHC_ForceClockOn (USDHC_Type * *base*, bool *enable*)
[inline], [static]**

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	enable/disable flag

**46.7.21 uint32_t USDHC_SetSdClock (USDHC_Type * *base*, uint32_t *srcClock_Hz*,
uint32_t *busClock_Hz*)**

Parameters

<i>base</i>	USDHC peripheral base address.
<i>srcClock_Hz</i>	USDHC 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 for SD bus.

46.7.22 bool USDHC_SetCardActive (USDHC_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

<i>base</i>	USDHC 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.

46.7.23 static void USDHC_AssertHardwareReset (USDHC_Type * *base*, bool *high*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>high</i>	1 or 0 level

46.7.24 static void USDHC_SetDataBusWidth (USDHC_Type * *base*, usdhc_data_bus_width_t *width*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>width</i>	Data transfer width.

46.7.25 static void USDHC_WriteData (USDHC_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>	USDHC peripheral base address.
<i>data</i>	The data about to be sent.

46.7.26 `static uint32_t USDHC_ReadData (USDHC_Type * base) [inline],
[static]`

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

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

Returns

The data has been read.

46.7.27 void USDHC_SendCommand (USDHC_Type * *base*, usdhc_command_t * *command*)

Parameters

<i>base</i>	USDHC peripheral base address.
<i>command</i>	configuration

46.7.28 static void USDHC_EnableWakeupEvent (USDHC_Type * *base*, uint32_t *mask*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	Wakeup events mask(_usdhc_wakeup_event).
<i>enable</i>	True to enable, false to disable.

46.7.29 static void USDHC_CardDetectByData3 (USDHC_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	enable/disable flag

46.7.30 static bool USDHC_DetectCardInsert (USDHC_Type * *base*) [inline], [static]

Parameters

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

46.7.31 `static void USDHC_EnableSdioControl (USDHC_Type * base, uint32_t mask, bool enable) [inline], [static]`

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	SDIO card control flags mask(<code>_usdhc_sdio_control_flag</code>).
<i>enable</i>	True to enable, false to disable.

46.7.32 `static void USDHC_SetContinueRequest (USDHC_Type * base) [inline], [static]`

Parameters

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

46.7.33 `static void USDHC_RequestStopAtBlockGap (USDHC_Type * base, bool enable) [inline], [static]`

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	True to stop at block gap, false to normal transfer.

46.7.34 `void USDHC_SetMmcBootConfig (USDHC_Type * base, const usdhc_boot_config_t * config)`

Example:

```
usdhc_boot_config_t config;
config.ackTimeoutCount = 4;
config.bootMode = kUSDHC_BootModeNormal;
config.blockCount = 5;
config.enableBootAck = true;
```

```
config.enableBoot = true;
config.enableAutoStopAtBlockGap = true;
USDHC_SetMmcBootConfig(USDHC, &config);
```

Parameters

<i>base</i>	USDHC peripheral base address.
<i>config</i>	The MMC boot configuration information.

46.7.35 static void USDHC_EnableMmcBoot (USDHC_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	True to enable, false to disable.

46.7.36 static void USDHC_SetForceEvent (USDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The force events bit position (_usdhc_force_event).

46.7.37 static void USDHC_SelectVoltage (USDHC_Type * *base*, bool *en18v*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>en18v</i>	True means 1.8V, false means 3.0V.

46.7.38 static bool USDHC_RequestTuningForSDR50 (USDHC_Type * *base*) [inline], [static]

When this bit set, application shall perform tuning for SDR50 mode.

Parameters

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

46.7.39 static bool USDHC_RequestReTuning (USDHC_Type * *base*) [inline], [static]

When this bit is set, user should do manual tuning or standard tuning function.

Parameters

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

46.7.40 static void USDHC_EnableAutoTuning (USDHC_Type * *base*, bool *enable*) [inline], [static]

This function should be called after tuning function execute pass, auto tuning will handle by hardware.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	enable/disable flag

46.7.41 void USDHC_EnableAutoTuningForCmdAndData (USDHC_Type * *base*)

Parameters

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

46.7.42 void USDHC_EnableManualTuning (USDHC_Type * *base*, bool *enable*)

User should handle the tuning cmd and find the boundary of the delay then calculate a average value which will be configured to the **CLK_TUNE_CTRL_STATUS** This function should be called before function [USDHC_AdjustDelayForManualTuning](#).

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	tuning enable flag

46.7.43 static uint32_t USDHC_GetTuningDelayStatus (USDHC_Type * *base*) [inline], [static]

Parameters

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

Return values

<i>CLK</i>	Tuning Control and Status register value.
------------	---

46.7.44 status_t USDHC_SetTuningDelay (USDHC_Type * *base*, uint32_t *preDelay*, uint32_t *outDelay*, uint32_t *postDelay*)

Parameters

<i>base</i>	USDHC peripheral base address.
<i>preDelay</i>	Set the number of delay cells on the feedback clock between the feedback clock and CLK_PRE.
<i>outDelay</i>	Set the number of delay cells on the feedback clock between CLK_PRE and CLK_OUT.
<i>postDelay</i>	Set the number of delay cells on the feedback clock between CLK_OUT and CLK_POST.

Return values

<i>kStatus_Fail</i>	config the delay setting fail
<i>kStatus_Success</i>	config the delay setting success

46.7.45 status_t USDHC_AdjustDelayForManualTuning (USDHC_Type * *base*, uint32_t *delay*)

Deprecated Do not use this function. It has been superseded by USDHC_SetTuningDelay

Parameters

<i>base</i>	USDHC peripheral base address.
<i>delay</i>	setting configuration

Return values

<i>kStatus_Fail</i>	config the delay setting fail
<i>kStatus_Success</i>	config the delay setting success

46.7.46 static void USDHC_SetStandardTuningCounter (USDHC_Type * *base*, uint8_t *counter*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>counter</i>	tuning counter

Return values

<i>kStatus_Fail</i>	config the delay setting fail
<i>kStatus_Success</i>	config the delay setting success

46.7.47 void USDHC_EnableStandardTuning (USDHC_Type * *base*, uint32_t *tuningStartTap*, uint32_t *step*, bool *enable*)

The standard tuning window and tuning counter using the default config tuning cmd is sent by the software, user need to check whether the tuning result can be used for SDR50, SDR104, and HS200 mode tuning.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>tuningStartTap</i>	start tap
<i>step</i>	tuning step

<i>enable</i>	enable/disable flag
---------------	---------------------

46.7.48 `static uint32_t USDHC_GetExecuteStdTuningStatus (USDHC_Type * base) [inline], [static]`

Parameters

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

46.7.49 `static uint32_t USDHC_CheckStdTuningResult (USDHC_Type * base) [inline], [static]`

Parameters

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

46.7.50 `static uint32_t USDHC_CheckTuningError (USDHC_Type * base) [inline], [static]`

Parameters

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

46.7.51 `void USDHC_EnableDDRMMode (USDHC_Type * base, bool enable, uint32_t nibblePos)`

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	enable/disable flag
<i>nibblePos</i>	nibble position

46.7.52 `static void USDHC_EnableHS400Mode (USDHC_Type * base, bool enable
) [inline], [static]`

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	enable/disable flag

46.7.53 `static void USDHC_ResetStrobeDLL (USDHC_Type * base) [inline], [static]`

Parameters

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

46.7.54 `static void USDHC_EnableStrobeDLL (USDHC_Type * base, bool enable) [inline], [static]`

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	enable/disable flag

46.7.55 `void USDHC_ConfigStrobeDLL (USDHC_Type * base, uint32_t delayTarget, uint32_t updateInterval)`

Parameters

<i>base</i>	USDHC peripheral base address.
<i>delayTarget</i>	delay target
<i>updateInterval</i>	update interval

46.7.56 `static void USDHC_SetStrobeDIIOverride (USDHC_Type * base, uint32_t delayTaps) [inline], [static]`

Parameters

<i>base</i>	USDHC peripheral base address.
<i>delayTaps</i>	Valid delay taps range from 1 - 128 taps. A value of 0 selects tap 1, and a value of 0x7F selects tap 128.

46.7.57 `static uint32_t USDHC_GetStrobeDLLStatus (USDHC_Type * base)
[inline], [static]`

Parameters

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

46.7.58 `void USDHC_SetDataConfig (USDHC_Type * base, usdhc_transfer-
_direction_t dataDirection, uint32_t blockCount, uint32_t blockSize
)`

Parameters

<i>base</i>	USDHC peripheral base address.
<i>dataDirection</i>	Data direction, tx or rx.
<i>blockCount</i>	Data block count.
<i>blockSize</i>	Data block size.

46.7.59 `void USDHC_TransferCreateHandle (USDHC_Type * base,
usdhc_handle_t * handle, const usdhc_transfer_callback_t * callback,
void * userData)`

Parameters

<i>base</i>	USDHC peripheral base address.
<i>handle</i>	USDHC handle pointer.

<i>callback</i>	Structure pointer to contain all callback functions.
<i>userData</i>	Callback function parameter.

46.7.60 **status_t USDHC_TransferNonBlocking (USDHC_Type * *base*, usdhc_handle_t * *handle*, usdhc_adma_config_t * *dmaConfig*, usdhc_transfer_t * *transfer*)**

This function sends a command and data and returns immediately. It doesn't wait for the transfer to complete or to encounter an error. 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 API [USDHC_TransferCreateHandle](#) when calling this API.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>handle</i>	USDHC handle.
<i>dmaConfig</i>	ADMA configuration.
<i>transfer</i>	Transfer content.

Return values

<i>kStatus_InvalidArgument</i>	Argument is invalid.
<i>kStatus_USDHC_Busy-Transferring</i>	Busy transferring.
<i>kStatus_USDHC_-PrepareAdmaDescriptor-Failed</i>	Prepare ADMA descriptor failed.
<i>kStatus_Success</i>	Operate successfully.

46.7.61 **status_t USDHC_TransferBlocking (USDHC_Type * *base*, usdhc_adma_config_t * *dmaConfig*, usdhc_transfer_t * *transfer*)**

This function waits until the command response/data is received or the USDHC encounters an error by polling the status flag.

The application must not call this API in multiple threads at the same time. Because this API doesn't support the re-entry mechanism.

Note

There is no need to call API [USDHC_TransferCreateHandle](#) when calling this API.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>dmaConfig</i>	adma configuration
<i>transfer</i>	Transfer content.

Return values

<i>kStatus_InvalidArgument</i>	Argument is invalid.
<i>kStatus_USDHC_-PrepareAdmaDescriptor-Failed</i>	Prepare ADMA descriptor failed.
<i>kStatus_USDHC_-SendCommandFailed</i>	Send command failed.
<i>kStatus_USDHC_-TransferDataFailed</i>	Transfer data failed.
<i>kStatus_Success</i>	Operate successfully.

46.7.62 void USDHC_TransferHandleIRQ (USDHC_Type * *base*, usdhc_handle_t * *handle*)

This function deals with the IRQs on the given host controller.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>handle</i>	USDHC handle.

Chapter 47

WDOG: Watchdog Timer Driver

47.1 Overview

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

47.2 Typical use case

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

Data Structures

- struct `wdog_work_mode_t`
Defines WDOG work mode. [More...](#)
- struct `wdog_config_t`
Describes WDOG configuration structure. [More...](#)

Enumerations

- enum `_wdog_interrupt_enable` { `kWDOG_InterruptEnable` = `WDOG_WICR_WIE_MASK` }
WDOG interrupt configuration structure, default settings all disabled.
- enum `_wdog_status_flags` {
`kWDOG_RunningFlag` = `WDOG_WCR_WDE_MASK`,
`kWDOG_PowerOnResetFlag` = `WDOG_WRSR_POR_MASK`,
`kWDOG_TimeoutResetFlag` = `WDOG_WRSR_TOUT_MASK`,
`kWDOG_SoftwareResetFlag` = `WDOG_WRSR_SFTW_MASK`,
`kWDOG_InterruptFlag` = `WDOG_WICR_WTIS_MASK` }
WDOG status flags.

Driver version

- `#define FSL_WDOG_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`
Defines WDOG driver version.

Refresh sequence

- `#define WDOG_REFRESH_KEY (0xAAAA5555U)`

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.*
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_TriggerSystemSoftwareReset](#) (WDOG_Type *base)
- *Trigger the system software reset.*
static void [WDOG_TriggerSoftwareSignal](#) (WDOG_Type *base)
- *Trigger an output assertion.*
static void [WDOG_EnableInterrupts](#) (WDOG_Type *base, uint16_t mask)
- *Enables the WDOG interrupt.*
uint16_t [WDOG_GetStatusFlags](#) (WDOG_Type *base)
- *Gets the WDOG all reset status flags.*
void [WDOG_ClearInterruptStatus](#) (WDOG_Type *base, uint16_t mask)
- *Clears the WDOG flag.*
static void [WDOG_SetTimeoutValue](#) (WDOG_Type *base, uint16_t timeoutCount)
- *Sets the WDOG timeout value.*
static void [WDOG_SetInterruptTimeoutValue](#) (WDOG_Type *base, uint16_t timeoutCount)
- *Sets the WDOG interrupt count timeout value.*
static void [WDOG_DisablePowerDownEnable](#) (WDOG_Type *base)
- *Disable the WDOG power down enable bit.*
void [WDOG_Refresh](#) (WDOG_Type *base)
- *Refreshes the WDOG timer.*

47.3 Data Structure Documentation

47.3.1 struct wdog_work_mode_t

Data Fields

- bool [enableWait](#)
continue or suspend WDOG in wait mode
- bool [enableStop](#)
continue or suspend WDOG in stop mode
- bool [enableDebug](#)
continue or suspend WDOG in debug mode

47.3.2 struct wdog_config_t

Data Fields

- bool [enableWdog](#)
Enables or disables WDOG.
- [wdog_work_mode_t](#) [workMode](#)
Configures WDOG work mode in debug stop and wait mode.
- bool [enableInterrupt](#)

- *Enables or disables WDOG interrupt.*
uint16_t `timeoutValue`
Timeout value.
- uint16_t `interruptTimeValue`
Interrupt count timeout value.
- bool `softwareResetExtension`
software reset extension
- bool `enablePowerDown`
power down enable bit
- bool `enableTimeOutAssert`
Enable WDOG_B timeout assertion.

Field Documentation

(1) bool `wdog_config_t::enableTimeOutAssert`

47.4 Enumeration Type Documentation

47.4.1 enum `_wdog_interrupt_enable`

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

Enumerator

kWDOG_InterruptEnable WDOG timeout generates an interrupt before reset.

47.4.2 enum `_wdog_status_flags`

This structure contains the WDOG status flags for use in the WDOG functions.

Enumerator

kWDOG_RunningFlag Running flag, set when WDOG is enabled.

kWDOG_PowerOnResetFlag Power On flag, set when reset is the result of a powerOnReset.

kWDOG_TimeoutResetFlag Timeout flag, set when reset is the result of a timeout.

kWDOG_SoftwareResetFlag Software flag, set when reset is the result of a software.

kWDOG_InterruptFlag interrupt flag, whether interrupt has occurred or not

47.5 Function Documentation

47.5.1 void `WDOG_GetDefaultConfig (wdog_config_t * config)`

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

```
* wdogConfig->enableWdog = true;
* wdogConfig->workMode.enableWait = true;
* wdogConfig->workMode.enableStop = false;
```

```

*  wdogConfig->workMode.enableDebug = false;
*  wdogConfig->enableInterrupt = false;
*  wdogConfig->enablePowerdown = false;
*  wdogConfig->resetExtension = false;
*  wdogConfig->timeoutValue = 0xFFU;
*  wdogConfig->interruptTimeValue = 0x04u;
*

```

Parameters

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

See Also

[wdog_config_t](#)

47.5.2 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.

This is an example.

```

*  wdog_config_t config;
*  WDOG_GetDefaultConfig(&config);
*  config.timeoutValue = 0xffU;
*  config->interruptTimeValue = 0x04u;
*  WDOG_Init(wdog_base, &config);
*

```

Parameters

<i>base</i>	WDOG peripheral base address
<i>config</i>	The configuration of WDOG

47.5.3 void WDOG_Deinit (WDOG_Type * *base*)

This function shuts down the WDOG. Watchdog Enable bit is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. This bit(WDE) can be set/reset only in debug mode(exception).

47.5.4 static void WDOG_Enable (WDOG_Type * *base*) [inline], [static]

This function writes a value into the WDOG_WCR register to enable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. only debug mode exception.

Parameters

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

47.5.5 static void WDOG_Disable (WDOG_Type * *base*) [inline], [static]

This function writes a value into the WDOG_WCR register to disable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. only debug mode exception

Parameters

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

47.5.6 static void WDOG_TriggerSystemSoftwareReset (WDOG_Type * *base*) [inline], [static]

This function will write to the WCR[SRS] bit to trigger a software system reset. This bit will automatically resets to "1" after it has been asserted to "0". Note: Calling this API will reset the system right now, please using it with more attention.

Parameters

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

47.5.7 static void WDOG_TriggerSoftwareSignal (WDOG_Type * *base*) [inline], [static]

This function will write to the WCR[WDA] bit to trigger WDOG_B signal assertion. The WDOG_B signal can be routed to external pin of the chip, the output pin will turn to assertion along with WDOG_B signal. Note: The WDOG_B signal will remain assert until a power on reset occurred, so, please take more attention while calling it.

Parameters

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

47.5.8 static void WDOG_EnableInterrupts (WDOG_Type * *base*, uint16_t *mask*)
[inline], [static]

This bit is a write once only bit. Once the software does a write access to this bit, it will get locked and cannot be reprogrammed until the next system reset assertion

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

47.5.9 uint16_t WDOG_GetStatusFlags (WDOG_Type * *base*)

This function gets all reset status flags.

```
* uint16_t status;
* status = WDOG_GetStatusFlags (wdog_base);
*
```

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](#)

- true: a related status flag has been set.
- false: a related status flag is not set.

47.5.10 void WDOG_ClearInterruptStatus (WDOG_Type * *base*, uint16_t *mask*)

This function clears the WDOG status flag.

This is an example for clearing the interrupt flag.

```
* WDOG_ClearStatusFlags (wdog_base, kWDOG_InterruptFlag);
*
```


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

47.5.11 **static void WDOG_SetTimeoutValue (WDOG_Type * *base*, uint16_t *timeoutCount*) [inline], [static]**

This function sets the timeout value. This function writes a value into WCR registers. The time-out value can be written at any point of time but it is loaded to the counter at the time when WDOG is enabled or after the service routine has been performed.

Parameters

<i>base</i>	WDOG peripheral base address
<i>timeoutCount</i>	WDOG timeout value; count of WDOG clock tick.

47.5.12 **static void WDOG_SetInterruptTimeoutValue (WDOG_Type * *base*, uint16_t *timeoutCount*) [inline], [static]**

This function sets the interrupt count timeout value. This function writes a value into WIC registers which are write-once. This field is write once only. Once the software does a write access to this field, it will get locked and cannot be reprogrammed until the next system reset assertion.

Parameters

<i>base</i>	WDOG peripheral base address
<i>timeoutCount</i>	WDOG timeout value; count of WDOG clock tick.

47.5.13 **static void WDOG_DisablePowerDownEnable (WDOG_Type * *base*) [inline], [static]**

This function disable the WDOG power down enable(PDE). This function writes a value into WMCR registers which are write-once. This field is write once only. Once software sets this bit it cannot be reset until the next system reset.

Parameters

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

47.5.14 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
-------------	------------------------------

Chapter 48

XBARA: Inter-Peripheral Crossbar Switch

48.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Peripheral Crossbar Switch (XBARA) block of MCUXpresso SDK devices.

The XBARA peripheral driver configures the XBARA (Inter-Peripheral Crossbar Switch) and handles initialization and configuration of the XBARA module.

XBARA driver has two parts:

- Signal connection interconnects input and output signals.
- Active edge feature - Some of the outputs provide an active edge detection. If an active edge occurs, an interrupt or a DMA request can be called. APIs handle user callbacks for the interrupts. The driver also includes API for clearing and reading status bits.

48.2 Function

48.2.1 XBARA Initialization

To initialize the XBARA driver, a state structure has to be passed into the initialization function. This block of memory keeps pointers to user's callback functions and parameters to these functions. The XBARA module is initialized by calling the [XBARA_Init\(\)](#) function.

48.2.2 Call diagram

1. Call the "XBARA_Init()" function to initialize the XBARA module.
2. Optionally, call the "XBARA_SetSignalsConnection()" function to Set connection between the selected XBARA_IN[*] input and the XBARA_OUT[*] output signal. It connects the XBARA input to the selected XBARA output. A configuration structure of the "xbara_input_signal_t" type and "xbara_output_signal_t" type is required.
3. Call the "XBARA_SetOutputSignalConfig" function to set the active edge features, such interrupts or DMA requests. A configuration structure of the "xbara_control_config_t" type is required to point to structure that keeps configuration of control register.
4. Finally, the XBARA works properly.

48.3 Typical use case

Data Structures

- struct [xbara_control_config_t](#)
Defines the configuration structure of the XBARA control register. [More...](#)

Enumerations

- enum `xbara_active_edge_t` {
`kXBARA_EdgeNone` = 0U,
`kXBARA_EdgeRising` = 1U,
`kXBARA_EdgeFalling` = 2U,
`kXBARA_EdgeRisingAndFalling` = 3U }
XBARA active edge for detection.
- enum `xbara_request_t` {
`kXBARA_RequestDisable` = 0U,
`kXBARA_RequestDMAEnable` = 1U,
`kXBARA_RequestInterruptEnable` = 2U }
Defines the XBARA DMA and interrupt configurations.
- enum `xbara_status_flag_t` {
`kXBARA_EdgeDetectionOut0`,
`kXBARA_EdgeDetectionOut1`,
`kXBARA_EdgeDetectionOut2`,
`kXBARA_EdgeDetectionOut3` }
XBARA status flags.

XBARA functional Operation.

- void `XBARA_Init` (`XBARA_Type *base`)
Initializes the XBARA module.
- void `XBARA_Deinit` (`XBARA_Type *base`)
Shuts down the XBARA module.
- void `XBARA_SetSignalsConnection` (`XBARA_Type *base`, `xbar_input_signal_t input`, `xbar_output_signal_t output`)
Sets a connection between the selected XBARA_IN[] input and the XBARA_OUT[*] output signal.*
- `uint32_t XBARA_GetStatusFlags` (`XBARA_Type *base`)
Gets the active edge detection status.
- void `XBARA_ClearStatusFlags` (`XBARA_Type *base`, `uint32_t mask`)
Clears the edge detection status flags of relative mask.
- void `XBARA_SetOutputSignalConfig` (`XBARA_Type *base`, `xbar_output_signal_t output`, `const xbara_control_config_t *controlConfig`)
Configures the XBARA control register.

48.4 Data Structure Documentation

48.4.1 struct `xbara_control_config_t`

This structure keeps the configuration of XBARA control register for one output. Control registers are available only for a few outputs. Not every XBARA module has control registers.

Data Fields

- `xbara_active_edge_t activeEdge`

- *Active edge to be detected.*
xbara_request_t requestType
Selects DMA/Interrupt request.

Field Documentation

- (1) **xbara_active_edge_t xbara_control_config_t::activeEdge**
- (2) **xbara_request_t xbara_control_config_t::requestType**

48.5 Enumeration Type Documentation

48.5.1 enum xbara_active_edge_t

Enumerator

- kXBARA_EdgeNone*** Edge detection status bit never asserts.
- kXBARA_EdgeRising*** Edge detection status bit asserts on rising edges.
- kXBARA_EdgeFalling*** Edge detection status bit asserts on falling edges.
- kXBARA_EdgeRisingAndFalling*** Edge detection status bit asserts on rising and falling edges.

48.5.2 enum xbara_request_t

Enumerator

- kXBARA_RequestDisable*** Interrupt and DMA are disabled.
- kXBARA_RequestDMAEnable*** DMA enabled, interrupt disabled.
- kXBARA_RequestInterruptEnable*** Interrupt enabled, DMA disabled.

48.5.3 enum xbara_status_flag_t

This provides constants for the XBARA status flags for use in the XBARA functions.

Enumerator

- kXBARA_EdgeDetectionOut0*** XBAR_OUT0 active edge interrupt flag, sets when active edge detected.
- kXBARA_EdgeDetectionOut1*** XBAR_OUT1 active edge interrupt flag, sets when active edge detected.
- kXBARA_EdgeDetectionOut2*** XBAR_OUT2 active edge interrupt flag, sets when active edge detected.
- kXBARA_EdgeDetectionOut3*** XBAR_OUT3 active edge interrupt flag, sets when active edge detected.

48.6 Function Documentation

48.6.1 void XBARA_Init (XBARA_Type * *base*)

This function un-gates the XBARA clock.

Parameters

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

48.6.2 void XBARA_Deinit (XBARA_Type * *base*)

This function disables XBARA clock.

Parameters

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

48.6.3 void XBARA_SetSignalsConnection (XBARA_Type * *base*, xbar_input_signal_t *input*, xbar_output_signal_t *output*)

This function connects the XBARA input to the selected XBARA output. If more than one XBARA module is available, only the inputs and outputs from the same module can be connected.

Example:

```
XBARA_SetSignalsConnection(XBARA, kXBARA_InputPIT_TRG0, kXBARA_OutputDMAMUX18);
```

Parameters

<i>base</i>	XBARA peripheral address.
<i>input</i>	XBARA input signal.
<i>output</i>	XBARA output signal.

48.6.4 uint32_t XBARA_GetStatusFlags (XBARA_Type * *base*)

This function gets the active edge detect status of all XBARA_OUTs. If the active edge occurs, the return value is asserted. When the interrupt or the DMA functionality is enabled for the XBARA_OUTx, this field is 1 when the interrupt or DMA request is asserted and 0 when the interrupt or DMA request has been cleared.

Parameters

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

Returns

the mask of these status flag bits.

48.6.5 void XBARA_ClearStatusFlags (XBARA_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	XBARA peripheral address.
<i>mask</i>	the status flags to clear.

48.6.6 void XBARA_SetOutputSignalConfig (XBARA_Type * *base*, xbar_output_signal_t *output*, const xbara_control_config_t * *controlConfig*)

This function configures an XBARA control register. The active edge detection and the DMA/IRQ function on the corresponding XBARA output can be set.

Example:

```
xbara_control_config_t userConfig;
userConfig.activeEdge = kXBARA_EdgeRising;
userConfig.requestType = kXBARA_RequestInterruptEnalbe;
XBARA_SetOutputSignalConfig(XBARA, kXBARA_OutputDMAMUX18, &userConfig);
```

Parameters

<i>base</i>	XBARA peripheral address.
<i>output</i>	XBARA output number.
<i>controlConfig</i>	Pointer to structure that keeps configuration of control register.

Chapter 49

XBARB: Inter-Peripheral Crossbar Switch

49.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Peripheral Crossbar Switch (XBARB) block of MCUXpresso SDK devices.

The XBARB peripheral driver configures the XBARB (Inter-Peripheral Crossbar Switch) and handles initialization and configuration of the XBARB module.

XBARB driver has two parts:

- Signal connection interconnects input and output signals.

49.2 Function groups

49.2.1 XBARB Initialization

To initialize the XBARB driver, a state structure has to be passed into the initialization function. This block of memory keeps pointers to user's callback functions and parameters to these functions. The XBARB module is initialized by calling the [XBARB_Init\(\)](#) function.

49.2.2 Call diagram

1. Call the "XBARB_Init()" function to initialize the XBARB module.
2. Optionally, call the "XBARB_SetSignalsConnection()" function to Set connection between the selected XBARB_IN[*] input and the XBARB_OUT[*] output signal. It connects the XBARB input to the selected XBARB output. A configuration structure of the "xbarb_input_signal_t" type and "xbarb_output_signal_t" type is required.
3. Finally, the XBARB works properly.

49.3 Typical use case

XBARB functional Operation.

- void [XBARB_Init](#) (XBARB_Type *base)
Initializes the XBARB module.
- void [XBARB_Deinit](#) (XBARB_Type *base)
Shuts down the XBARB module.
- void [XBARB_SetSignalsConnection](#) (XBARB_Type *base, xbarb_input_signal_t input, xbarb_output_signal_t output)
Configures a connection between the selected XBARB_IN[] input and the XBARB_OUT[*] output signal.*

49.4 Function Documentation

49.4.1 void XBARB_Init (XBARB_Type * *base*)

This function un-gates the XBARB clock.

Parameters

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

49.4.2 void XBARB_Deinit (XBARB_Type * *base*)

This function disables XBARB clock.

Parameters

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

49.4.3 void XBARB_SetSignalsConnection (XBARB_Type * *base*, xbar_input_signal_t *input*, xbar_output_signal_t *output*)

This function configures which XBARB input is connected to the selected XBARB output. If more than one XBARB module is available, only the inputs and outputs from the same module can be connected.

Parameters

<i>base</i>	XBARB peripheral address.
<i>input</i>	XBARB input signal.
<i>output</i>	XBARB output signal.

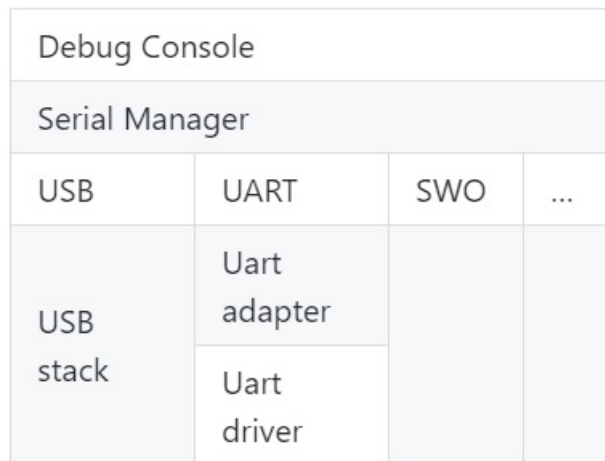
Chapter 50

Debug Console

50.1 Overview

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

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



Debug console overview

50.2 Function groups

50.2.1 Initialization

To initialize the debug console, call the `DbgConsole_Init()` function with these parameters. This function automatically enables the module and the clock.

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

Select the supported debug console hardware device type, such as

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

After the initialization is successful, stdout and stdin are connected to the selected peripheral. This example shows how to call the `DbgConsole_Init()` given the user configuration structure.

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

50.2.2 Advanced Feature

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

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

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

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

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

length	Description
Do not support	

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

specifier	Description
-----------	-------------

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

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

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

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

specifier	Qualifying Input	Type of argument
c	Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end.	char *
i	Integer: : Number optionally preceded with a + or - sign	int *
d	Decimal integer: Number optionally preceded with a + or - sign	int *
a, A, e, E, f, F, g, G	Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4	float *
o	Octal Integer:	int *
s	String of characters. This reads subsequent characters until a white space is found (white space characters are considered to be blank, newline, and tab).	char *
u	Unsigned decimal integer.	unsigned int *

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

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

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

```
#if SDK_DEBUGCONSOLE == DEBUGCONSOLE_DISABLE /* Disable debug console */
#define PRINTF
#define SCANF
#define PUTCHAR
#define GETCHAR
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_SDK /* Select printf, scanf, putchar, getchar of SDK
```



```

        version. */
#define PRINTF DbgConsole_Printf
#define SCANF DbgConsole_Scanf
#define PUTCHAR DbgConsole_Putchar
#define GETCHAR DbgConsole_Getchar
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN /* Select printf, scanf, putchar, getchar of
    toolchain. */
#define PRINTF printf
#define SCANF scanf
#define PUTCHAR putchar
#define GETCHAR getchar
#endif /* SDK_DEBUGCONSOLE */

```

50.2.3 SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART

There are two macros `SDK_DEBUGCONSOLE` and `SDK_DEBUGCONSOLE_UART` added to configure `PRINTF` and low level output perihperal.

- The macro `SDK_DEBUGCONSOLE` is used for forntend. Whether debug console redirect to toolchain or SDK or disabled, it decides which is the frontend of the debug console, Tool chain or SDK. The fuction can be set by the macro `SDK_DEBUGCONSOLE`.
- The macro `SDK_DEBUGCONSOLE_UART` is used for backend. It is use to decide whether provide low level IO implementation to toolchain `printf` and `scanf`. For example, within MCU-Xpresso, 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. The macro does not specifically refer to the perihperal "UART". It refers to the external perihperal similar to UART, like as USB CDC, UART, SWO, etc. So if the macro `SDK_DEBUGCONSOLE_UART` is not defined when tool-chain `printf` is calling, the semihosting will be used.

The following the matrix show the effects of `SDK_DEBUGCONSOLE` and `SDK_DEBUGCONSOLE_UART` on `PRINTF` and `printf`. The green mark is the default setting of the debug console.

<code>SDK_DEBUGCONSOLE</code>	<code>SDK_DEBUGCONSOLE_UART</code>	<code>PRINTF</code>	<code>printf</code>
<code>DEBUGCONSOLE_-REDIRECT_TO_SDK</code>	defined	Low level peripheral*	Low level peripheral
<code>DEBUGCONSOLE_-REDIRECT_TO_SDK</code>	undefined	Low level peripheral*	semihost
<code>DEBUGCONSOLE_-REDIRECT_TO_TO-OLCHAIN</code>	defined	Low level peripheral*	Low level peripheral
<code>DEBUGCONSOLE_-REDIRECT_TO_TO-OLCHAIN</code>	undefined	semihost	semihost
<code>DEBUGCONSOLE_-DISABLE</code>	defined	No ouput	Low level peripheral
<code>DEBUGCONSOLE_-DISABLE</code>	undefined	No ouput	semihost

SDK_DEBUGCONSOLE	SDK_DEBUGCONSOLE_UART	PRINTF	printf
------------------	-----------------------	--------	--------

* the low level peripheral could be USB CDC, UART, or SWO, and so on.

50.3 Typical use case

Some examples use the PUTCHAR & GETCHAR function

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

Some examples use the PRINTF function

Statement prints the string format.

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

Statement prints the hexadecimal format/

```
PRINTF("0x%02X hexadecimal number 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\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](#)
- [debug console configuration](#)

The configuration is used for debug console only.

Macros

- #define [DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN](#) 0U
Definition select redirect toolchain printf, scanf to uart or not.
- #define [DEBUGCONSOLE_REDIRECT_TO_SDK](#) 1U
Select SDK version printf, scanf.
- #define [DEBUGCONSOLE_DISABLE](#) 2U
Disable debugconsole function.
- #define [SDK_DEBUGCONSOLE](#) [DEBUGCONSOLE_REDIRECT_TO_SDK](#)
Definition to select sdk or toolchain printf, scanf.
- #define [PRINTF](#) [DbgConsole_Printf](#)
Definition to select redirect toolchain printf, scanf to uart or not.

Variables

- [serial_handle_t g_serialHandle](#)
serial manager handle

Initialization

- [status_t DbgConsole_Init](#) (uint8_t instance, uint32_t baudRate, [serial_port_type_t](#) device, uint32_t clkSrcFreq)
Initializes the peripheral used for debug messages.
- [status_t DbgConsole_Deinit](#) (void)
De-initializes the peripheral used for debug messages.
- [status_t DbgConsole_EnterLowpower](#) (void)
Prepares to enter low power consumption.
- [status_t DbgConsole_ExitLowpower](#) (void)
Restores from low power consumption.
- int [DbgConsole_Printf](#) (const char *fmt_s,...)
Writes formatted output to the standard output stream.
- int [DbgConsole_Vprintf](#) (const char *fmt_s, va_list formatStringArg)
Writes formatted output to the standard output stream.
- int [DbgConsole_Putchar](#) (int ch)
Writes a character to stdout.
- int [DbgConsole_Scanf](#) (char *fmt_s,...)
Reads formatted data from the standard input stream.
- int [DbgConsole_Getchar](#) (void)

- *Reads a character from standard input.*
int [DbgConsole_BlockingPrintf](#) (const char *fmt_s,...)
Writes formatted output to the standard output stream with the blocking mode.
- int [DbgConsole_BlockingVprintf](#) (const char *fmt_s, va_list formatStringArg)
Writes formatted output to the standard output stream with the blocking mode.
- [status_t DbgConsole_Flush](#) (void)
Debug console flush.
- [status_t DbgConsole_TryGetchar](#) (char *ch)
Debug console try to get char This function provides a API which will not block current task, if character is available return it, otherwise return fail.

50.4 Macro Definition Documentation

50.4.1 #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U

Select toolchain printf and scanf.

50.4.2 #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U

50.4.3 #define DEBUGCONSOLE_DISABLE 2U

50.4.4 #define SDK_DEBUGCONSOLE DEBUGCONSOLE_REDIRECT_TO_SDK

The macro only support to be redefined in project setting.

50.4.5 #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.

50.5 Function Documentation

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

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

Parameters

<i>instance</i>	The instance of the module.If the device is kSerialPort_Uart, the instance is UART peripheral instance. The UART hardware peripheral type is determined by UART adapter. For example, if the instance is 1, if the lpuart_adapter.c is added to the current project, the UART peripheral is LPUART1. If the uart_adapter.c is added to the current project, the UART peripheral is UART1.
<i>baudRate</i>	The desired baud rate in bits per second.
<i>device</i>	Low level device type for the debug console, can be one of the following. <ul style="list-style-type: none"> • kSerialPort_Uart, • kSerialPort_UsbCdc
<i>clkSrcFreq</i>	Frequency of peripheral source clock.

Returns

Indicates whether initialization was successful or not.

Return values

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

50.5.2 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.

50.5.3 status_t DbgConsole_EnterLowpower (void)

This function is used to prepare to enter low power consumption.

Returns

Indicates whether de-initialization was successful or not.

50.5.4 status_t DbgConsole_ExitLowpower (void)

This function is used to restore from low power consumption.

Returns

Indicates whether de-initialization was successful or not.

50.5.5 int DbgConsole_Printf (const char * *fmt_s*, ...)

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

Parameters

<i>fmt_s</i>	Format control string.
--------------	------------------------

Returns

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

50.5.6 int DbgConsole_Vprintf (const char * *fmt_s*, va_list *formatStringArg*)

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

Parameters

<i>fmt_s</i>	Format control string.
<i>formatString-Arg</i>	Format arguments.

Returns

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

50.5.7 int DbgConsole_Putchar (int *ch*)

Call this function to write a character to stdout.

Parameters

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

Returns

Returns the character written.

50.5.8 int DbgConsole_Scanf (char * *fmt_s*, ...)

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

Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the `DEBUG_CONSOLE_TRANSFER_NON_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>fmt_s</i>	Format control string.
--------------	------------------------

Returns

Returns the number of fields successfully converted and assigned.

50.5.9 int DbgConsole_Getchar (void)

Call this function to read a character from standard input.

Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the `DEBUG_CONSOLE_TRANSFER_NON_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.

50.5.10 int DbgConsole_BlockingPrintf (const char * *fmt_s*, ...)

Call this function to write a formatted output to the standard output stream with the blocking mode. The function will send data with blocking mode no matter the `DEBUG_CONSOLE_TRANSFER_NON_BLOCKING` set or not. The function could be used in system ISR mode with `DEBUG_CONSOLE_TRANSFER_NON_BLOCKING` set.

Parameters

<i>fmt_s</i>	Format control string.
--------------	------------------------

Returns

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

50.5.11 int DbgConsole_BlockingVprintf (const char * *fmt_s*, va_list *formatStringArg*)

Call this function to write a formatted output to the standard output stream with the blocking mode. The function will send data with blocking mode no matter the `DEBUG_CONSOLE_TRANSFER_NON_BLOCKING` set or not. The function could be used in system ISR mode with `DEBUG_CONSOLE_TRANSFER_NON_BLOCKING` set.

Parameters

<i>fmt_s</i>	Format control string.
<i>formatStringArg</i>	Format arguments.

Returns

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

50.5.12 status_t DbgConsole_Flush (void)

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

Returns

Indicates whether wait idle was successful or not.

50.5.13 `status_t DbgConsole_TryGetchar (char * ch)`

Parameters

<i>ch</i>	the address of char to receive
-----------	--------------------------------

Returns

Indicates get char was successful or not.

50.6 debug console configuration

The configuration is used for debug console only.

50.6.1 Overview

Please note, it is not sued for debug console lite.

Macros

- #define **DEBUG_CONSOLE_TRANSMIT_BUFFER_LEN** (512U)
If Non-blocking mode is needed, please define it at project setting, otherwise blocking mode is the default transfer mode.
- #define **DEBUG_CONSOLE_RECEIVE_BUFFER_LEN** (1024U)
define the receive buffer length which is used to store the user input, buffer is enabled automatically when non-blocking transfer is using, This value will affect the RAM's utilization, should be set per paltform's capability and software requirement.
- #define **DEBUG_CONSOLE_TX_RELIABLE_ENABLE** (1U)
Whether enable the reliable TX function If the macro is zero, the reliable TX function of the debug console is disabled.
- #define **DEBUG_CONSOLE_RX_ENABLE** (1U)
Whether enable the RX function If the macro is zero, the receive function of the debug console is disabled.
- #define **DEBUG_CONSOLE_PRINTF_MAX_LOG_LEN** (128U)
define the MAX log length debug console support , that is when you call printf("log", x);, the log length can not bigger than this value.
- #define **DEBUG_CONSOLE_SCANF_MAX_LOG_LEN** (20U)
define the buffer support buffer scanf log length, that is when you call scanf("log", &x);, the log length can not bigger than this value.
- #define **DEBUG_CONSOLE_SYNCHRONIZATION_BM** 0
Debug console synchronization User should not change these macro for synchronization mode, but add the corresponding synchronization mechanism per different software environment.
- #define **DEBUG_CONSOLE_SYNCHRONIZATION_FREERTOS** 1
synchronization for freertos software
- #define **DEBUG_CONSOLE_SYNCHRONIZATION_MODE** **DEBUG_CONSOLE_SYNCHRONIZATION_BM**
RTOS synchronization mechanism disable If not defined, default is enable, to avoid multitask log print mess.
- #define **DEBUG_CONSOLE_ENABLE_ECHO_FUNCTION** 0
*echo function support If you want to use the echo function,please define **DEBUG_CONSOLE_ENABLE_ECHO** at your project setting.*
- #define **BOARD_USE_VIRTUALCOM** 0U
Definition to select virtual com(USB CDC) as the debug console.

50.6.2 Macro Definition Documentation

50.6.2.1 #define DEBUG_CONSOLE_TRANSMIT_BUFFER_LEN (512U)

Warning: If you want to use non-blocking transfer, please make sure the corresponding IO interrupt is enable, otherwise there is no output. And non-blocking is combine with buffer, no matter bare-metal or rtos. Below shows how to configure in your project if you want to use non-blocking mode. For IAR, right click project and select "Options", define it in "C/C++ Compiler->Preprocessor->Defined symbols". For KEIL, click "Options for Target... ", define it in "C/C++->Preprocessor Symbols->Define". For ARM-GCC, open CmakeLists.txt and add the following lines, "SET(CMAKE_C_FLAGS_DEBUG "\${CMAKE_C_FLAGS_DEBUG} -DDEBUG_CONSOLE_TRANSFER_NON_BLOCKING)" for debug target. "SET(CMAKE_C_FLAGS_RELEASE "\${CMAKE_C_FLAGS_RELEASE} -DDEBUG_CONSOLE_TRANSFER_NON_BLOCKING)" for release target. For MCUXpresso, right click project and select "Properties", define it in "C/C++ Build->Settings->MCU C Compiler->Preprocessor".

define the transmit buffer length which is used to store the multi task log, buffer is enabled automatically when non-blocking transfer is using, This value will affect the RAM's utilization, should be set per paltform's capability and software requirement. If it is configured too small, log maybe missed , because the log will not be buffered if the buffer is full, and the print will return immediately with -1. And this value should be multiple of 4 to meet memory alignment.

50.6.2.2 #define DEBUG_CONSOLE_RECEIVE_BUFFER_LEN (1024U)

If it is configured too small, log maybe missed, because buffer will be overwritten if buffer is too small. And this value should be multiple of 4 to meet memory alignment.

50.6.2.3 #define DEBUG_CONSOLE_TX_RELIABLE_ENABLE (1U)

When the macro is zero, the string of PRINTF will be thrown away after the transmit buffer is full.

50.6.2.4 #define DEBUG_CONSOLE_PRINTF_MAX_LOG_LEN (128U)

This macro decide the local log buffer length, the buffer locate at stack, the stack maybe overflow if the buffer is too big and current task stack size not big enough.

50.6.2.5 #define DEBUG_CONSOLE_SCANF_MAX_LOG_LEN (20U)

As same as the DEBUG_CONSOLE_BUFFER_PRINTF_MAX_LOG_LEN.

50.6.2.6 **#define DEBUG_CONSOLE_SYNCHRONIZATION_BM 0**

Such as, if another RTOS is used, add: `#define DEBUG_CONSOLE_SYNCHRONIZATION_XXXX 3` in this configuration file and implement the synchronization in `fsl.log.c`.

synchronization for baremetal software

50.6.2.7 **#define DEBUG_CONSOLE_SYNCHRONIZATION_MODE DEBUG_CONSOLE_SYNCHRONIZATION_BM**

If other RTOS is used, you can implement the RTOS's specific synchronization mechanism in `fsl.log.c`. If synchronization is disabled, log maybe messed on terminal.

50.6.2.8 **#define BOARD_USE_VIRTUALCOM 0U**

50.7 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.

50.7.1 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.

50.7.2 Guide Semihosting for Keil μ Vision

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

50.7.3 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.

50.7.4 Guide Semihosting for ARMGCC

Step 1: Setting up the environment

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

Add to "CMakeLists.txt"

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

Step 2: Building the project

1. Change "CMakeLists.txt":

Change "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "\${CMAKE_EXE_LINKER_FLAGS_RELEASE} -specs=nano.specs")"

to "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "\${CMAKE_EXE_LINKER_FLAGS_RELEASE} -specs=rdimon.specs")"

Replace paragraph

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-common")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffunction-sections")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fdata-sections")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffreestanding")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-builtin")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mthumb")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mapcs")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} --gc-sections")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -static")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -z")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} muldefs")

To

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} --specs=rdimon.specs ")

Remove

target_link_libraries(semihosting_ARMGCC.elf debug nosys)

2. Run "build_debug.bat" to build project

Step 3: Starting semihosting

1. Download the image and set as follows.

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

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

50.8 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.

50.8.1 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

50.8.1.1 Guide SWO for IAR

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

Step 1: Setting up the environment

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

NOTE: Case a or c only apply at example which enable swo function,the SDK_DEBUGCONSOLE_UART definition in fsl_debug_console.h. For case a and c, Do and not do the above third step will be not affect function.

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

Step 2: Building the project

Step 3: Starting swo

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

50.8.2 Guide SWO for Keil μ Vision

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.

50.8.3 Guide SWO for MCUXpresso IDE

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

50.8.4 Guide SWO for ARMGCC

NOTE: ARMGCC has no library support SWO.

Chapter 51

Notification Framework

51.1 Overview

This section describes the programming interface of the Notifier driver.

51.2 Notifier Overview

The Notifier provides a configuration dynamic change service. Based on this service, applications can switch between pre-defined configurations. The Notifier enables drivers and applications to register callback functions to this framework. Each time that the configuration is changed, drivers and applications receive a notification and change their settings. To simplify, the Notifier only supports the static callback registration. This means that, for applications, all callback functions are collected into a static table and passed to the Notifier.

These are the steps for the configuration transition.

1. Before configuration transition, the Notifier sends a "BEFORE" message to the callback table. When this message is received, IP drivers should check whether any current processes can be stopped and stop them. If the processes cannot be stopped, the callback function returns an error.
The Notifier supports two types of transition policies, a graceful policy and a forceful policy. When the graceful policy is used, if some callbacks return an error while sending a "BEFORE" message, the configuration transition stops and the Notifier sends a "RECOVER" message to all drivers that have stopped. Then, these drivers can recover the previous status and continue to work. When the forceful policy is used, drivers are stopped forcefully.
2. After the "BEFORE" message is processed successfully, the system switches to the new configuration.
3. After the configuration changes, the Notifier sends an "AFTER" message to the callback table to notify drivers that the configuration transition is finished.

This example shows how to use the Notifier in the Power Manager application.

```
#include "fsl_notifier.h"

// Definition of the Power Manager callback.
status_t callback0(notifier_notification_block_t *notify, void *data)
{
    status_t ret = kStatus_Success;

    ...
    ...
    ...

    return ret;
}

// Definition of the Power Manager user function.
status_t APP_PowerModeSwitch(notifier_user_config_t *targetConfig, void *
    userData)
```

```

{
    ...
    ...
    ...
}
...
...
...
...
...
// Main function.
int main(void)
{
    // Define a notifier handle.
    notifier_handle_t powerModeHandle;

    // Callback configuration.
    user_callback_data_t callbackData0;

    notifier_callback_config_t callbackCfg0 = {callback0,
        kNOTIFIER_CallbackBeforeAfter,
        (void *)&callbackData0};

    notifier_callback_config_t callbacks[] = {callbackCfg0};

    // Power mode configurations.
    power_user_config_t vlprConfig;
    power_user_config_t stopConfig;

    notifier_user_config_t *powerConfigs[] = {&vlprConfig, &stopConfig};

    // Definition of a transition to and out the power modes.
    vlprConfig.mode = kAPP_PowerModeVlpr;
    vlprConfig.enableLowPowerWakeUpOnInterrupt = false;

    stopConfig = vlprConfig;
    stopConfig.mode = kAPP_PowerModeStop;

    // Create Notifier handle.
    NOTIFIER_CreateHandle(&powerModeHandle, powerConfigs, 2U, callbacks, 1U,
        APP_PowerModeSwitch, NULL);
    ...
    ...
    // Power mode switch.
    NOTIFIER_switchConfig(&powerModeHandle, targetConfigIndex,
        kNOTIFIER_PolicyAgreement);
}

```

Data Structures

- struct [notifier_notification_block_t](#)
notification block passed to the registered callback function. *More...*
- struct [notifier_callback_config_t](#)
Callback configuration structure. *More...*
- struct [notifier_handle_t](#)
Notifier handle structure. *More...*

Typedefs

- typedef void [notifier_user_config_t](#)
Notifier user configuration type.
- typedef [status_t](#)(* [notifier_user_function_t](#))([notifier_user_config_t](#) *targetConfig, void *userData)

- Notifier user function prototype Use this function to execute specific operations in configuration switch.*
- typedef `status_t(* notifier_callback_t)(notifier_notification_block_t *notify, void *data)`
Callback prototype.

Enumerations

- enum `_notifier_status` {
 `kStatus_NOTIFIER_ErrorNotificationBefore`,
 `kStatus_NOTIFIER_ErrorNotificationAfter` }
Notifier error codes.
- enum `notifier_policy_t` {
 `kNOTIFIER_PolicyAgreement`,
 `kNOTIFIER_PolicyForcible` }
Notifier policies.
- enum `notifier_notification_type_t` {
 `kNOTIFIER_NotifyRecover` = 0x00U,
 `kNOTIFIER_NotifyBefore` = 0x01U,
 `kNOTIFIER_NotifyAfter` = 0x02U }
Notification type.
- enum `notifier_callback_type_t` {
 `kNOTIFIER_CallbackBefore` = 0x01U,
 `kNOTIFIER_CallbackAfter` = 0x02U,
 `kNOTIFIER_CallbackBeforeAfter` = 0x03U }
The callback type, which indicates kinds of notification the callback handles.

Functions

- `status_t NOTIFIER_CreateHandle` (`notifier_handle_t *notifierHandle`, `notifier_user_config_t **configs`, `uint8_t configsNumber`, `notifier_callback_config_t *callbacks`, `uint8_t callbacksNumber`, `notifier_user_function_t userFunction`, `void *userData`)
Creates a Notifier handle.
- `status_t NOTIFIER_SwitchConfig` (`notifier_handle_t *notifierHandle`, `uint8_t configIndex`, `notifier_policy_t policy`)
Switches the configuration according to a pre-defined structure.
- `uint8_t NOTIFIER_GetErrorCallbackIndex` (`notifier_handle_t *notifierHandle`)
This function returns the last failed notification callback.

51.3 Data Structure Documentation

51.3.1 struct `notifier_notification_block_t`

Data Fields

- `notifier_user_config_t * targetConfig`
Pointer to target configuration.
- `notifier_policy_t policy`
Configure transition policy.
- `notifier_notification_type_t notifyType`

Configure notification type.

Field Documentation

- (1) `notifier_user_config_t* notifier_notification_block_t::targetConfig`
- (2) `notifier_policy_t notifier_notification_block_t::policy`
- (3) `notifier_notification_type_t notifier_notification_block_t::notifyType`

51.3.2 struct `notifier_callback_config_t`

This structure holds the configuration of callbacks. Callbacks of this type are expected to be statically allocated. This structure contains the following application-defined data. `callback` - pointer to the callback function `callbackType` - specifies when the callback is called `callbackData` - pointer to the data passed to the callback.

Data Fields

- [notifier_callback_t callback](#)
Pointer to the callback function.
- [notifier_callback_type_t callbackType](#)
Callback type.
- `void * callbackData`
Pointer to the data passed to the callback.

Field Documentation

- (1) `notifier_callback_t notifier_callback_config_t::callback`
- (2) `notifier_callback_type_t notifier_callback_config_t::callbackType`
- (3) `void* notifier_callback_config_t::callbackData`

51.3.3 struct `notifier_handle_t`

Notifier handle structure. Contains data necessary for the Notifier proper function. Stores references to registered configurations, callbacks, information about their numbers, user function, user data, and other internal data. `NOTIFIER_CreateHandle()` must be called to initialize this handle.

Data Fields

- [notifier_user_config_t ** configsTable](#)
Pointer to configure table.
- `uint8_t configsNumber`
Number of configurations.

- [notifier_callback_config_t](#) * [callbacksTable](#)
Pointer to callback table.
- [uint8_t](#) [callbacksNumber](#)
Maximum number of callback configurations.
- [uint8_t](#) [errorCallbackIndex](#)
Index of callback returns error.
- [uint8_t](#) [currentConfigIndex](#)
Index of current configuration.
- [notifier_user_function_t](#) [userFunction](#)
User function.
- [void](#) * [userData](#)
User data passed to user function.

Field Documentation

- (1) [notifier_user_config_t](#)*[notifier_handle_t::configsTable](#)
- (2) [uint8_t](#) [notifier_handle_t::configsNumber](#)
- (3) [notifier_callback_config_t](#)*[notifier_handle_t::callbacksTable](#)
- (4) [uint8_t](#) [notifier_handle_t::callbacksNumber](#)
- (5) [uint8_t](#) [notifier_handle_t::errorCallbackIndex](#)
- (6) [uint8_t](#) [notifier_handle_t::currentConfigIndex](#)
- (7) [notifier_user_function_t](#) [notifier_handle_t::userFunction](#)
- (8) [void](#)* [notifier_handle_t::userData](#)

51.4 Typedef Documentation

51.4.1 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.

51.4.2 typedef [status_t](#)(* [notifier_user_function_t](#))([notifier_user_config_t](#) *[targetConfig](#), [void](#) *[userData](#))

Before and after this function execution, different notification is sent to registered callbacks. If this function returns any error code, [NOTIFIER_SwitchConfig\(\)](#) exits.

Parameters

<i>targetConfig</i>	target Configuration.
<i>userData</i>	Refers to other specific data passed to user function.

Returns

An error code or `kStatus_Success`.

51.4.3 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`.

51.5 Enumeration Type Documentation

51.5.1 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.

51.5.2 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.

51.5.3 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.

51.5.4 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.

51.6 Function Documentation

51.6.1 `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.

51.6.2 **status_t NOTIFIER_SwitchConfig (notifier_handle_t * *notifierHandle*, uint8_t *configIndex*, notifier_policy_t *policy*)**

This function sets the system to the target configuration. Before transition, the Notifier sends notifications to all callbacks registered to the callback table. Callbacks are invoked in the following order: All registered callbacks are notified ordered by index in the callbacks array. The same order is used for before and after switch notifications. The notifications before the configuration switch can be used to obtain confirmation about the change from registered callbacks. If any registered callback denies the configuration change, further execution of this function depends on the notifier policy: the configuration change is either forced (kNOTIFIER_PolicyForcible) or exited (kNOTIFIER_PolicyAgreement). When configuration change is forced, the result of the before switch notifications are ignored. If an agreement is required, if any callback returns an error code, further notifications before switch notifications are cancelled and all already notified callbacks are re-invoked. The index of the callback which returned error code during pre-switch notifications is stored (any error codes during callbacks re-invocation are ignored) and NOTIFIER_GetErrorCallback() can be used to get it. Regardless of the policies, if any callback returns an error code, an error code indicating in which phase the error occurred is returned when [NOTIFIER_SwitchConfig\(\)](#) exits.

Parameters

<i>notifierHandle</i>	pointer to notifier handle
<i>configIndex</i>	Index of the target configuration.
<i>policy</i>	Transaction policy, kNOTIFIER_PolicyAgreement or kNOTIFIER_PolicyForcible.

Returns

An error code or kStatus_Success.

51.6.3 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 52

Shell

52.1 Overview

This section describes the programming interface of the Shell middleware.

Shell controls MCUs by commands via the specified communication peripheral based on the debug console driver.

52.2 Function groups

52.2.1 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>");
```

52.2.2 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.

52.2.3 Shell Operation

```
SHELL_Init(s_shellHandle, s_serialHandle, "Test@SHELL>");  
SHELL_Task((s_shellHandle);
```

Data Structures

- struct `shell_command_t`
User command data configuration structure. [More...](#)

Macros

- #define `SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE`
Whether use non-blocking mode.
- #define `SHELL_AUTO_COMPLETE` (1U)
Macro to set on/off auto-complete feature.
- #define `SHELL_BUFFER_SIZE` (64U)
Macro to set console buffer size.
- #define `SHELL_MAX_ARGS` (8U)
Macro to set maximum arguments in command.
- #define `SHELL_HISTORY_COUNT` (3U)
Macro to set maximum count of history commands.
- #define `SHELL_IGNORE_PARAMETER_COUNT` (0xFF)
Macro to bypass arguments check.
- #define `SHELL_HANDLE_SIZE`
The handle size of the shell module.
- #define `SHELL_USE_COMMON_TASK` (0U)
Macro to determine whether use common task.
- #define `SHELL_TASK_PRIORITY` (2U)
Macro to set shell task priority.
- #define `SHELL_TASK_STACK_SIZE` (1000U)
Macro to set shell task stack size.
- #define `SHELL_HANDLE_DEFINE`(name) uint32_t name[((`SHELL_HANDLE_SIZE` + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
Defines the shell handle.
- #define `SHELL_COMMAND_DEFINE`(command, descriptor, callback, 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)`,
`kStatus_SHELL_RetUsage` = `MAKE_STATUS(kStatusGroup_SHELL, 4)` }
Shell status.

Shell functional operation

- `shell_status_t SHELL_Init` (`shell_handle_t` shellHandle, `serial_handle_t` serialHandle, `char *prompt`)
Initializes the shell module.
- `shell_status_t SHELL_RegisterCommand` (`shell_handle_t` shellHandle, `shell_command_t *shellCommand`)
Registers the shell command.
- `shell_status_t SHELL_UnregisterCommand` (`shell_command_t *shellCommand`)
Unregisters the shell command.
- `shell_status_t SHELL_Write` (`shell_handle_t` shellHandle, `const char *buffer`, `uint32_t length`)
Sends data to the shell output stream.
- `int SHELL_Printf` (`shell_handle_t` shellHandle, `const char *formatString,...`)
Writes formatted output to the shell output stream.
- `shell_status_t SHELL_WriteSynchronization` (`shell_handle_t` shellHandle, `const char *buffer`, `uint32_t length`)
Sends data to the shell output stream with OS synchronization.
- `int SHELL_PrintfSynchronization` (`shell_handle_t` shellHandle, `const char *formatString,...`)
Writes formatted output to the shell output stream with OS synchronization.
- `void SHELL_ChangePrompt` (`shell_handle_t` shellHandle, `char *prompt`)
Change shell prompt.
- `void SHELL_PrintPrompt` (`shell_handle_t` shellHandle)
Print shell prompt.
- `void SHELL_Task` (`shell_handle_t` shellHandle)
The task function for Shell.
- `static bool SHELL_checkRunningInIsr` (`void`)
Check if code is running in ISR.

52.3 Data Structure Documentation

52.3.1 struct shell_command_t

Data Fields

- `const char * pcCommand`
The command that is executed.
- `char * pcHelpString`
String that describes how to use the command.
- `const cmd_function_t pFuncCallBack`
A pointer to the callback function that returns the output generated by the command.
- `uint8_t cExpectedNumberOfParameters`
Commands expect a fixed number of parameters, which may be zero.
- `list_element_t link`
link of the element

Field Documentation

(1) `const char* shell_command_t::pcCommand`

For example "help". It must be all lower case.

(2) char* shell_command_t::pcHelpString

It should start with the command itself, and end with "\r\n". For example "help: Returns a list of all the commands\r\n".

(3) const cmd_function_t shell_command_t::pFuncCallback**(4) uint8_t shell_command_t::cExpectedNumberOfParameters****52.4 Macro Definition Documentation****52.4.1 #define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE****52.4.2 #define SHELL_AUTO_COMPLETE (1U)****52.4.3 #define SHELL_BUFFER_SIZE (64U)****52.4.4 #define SHELL_MAX_ARGS (8U)****52.4.5 #define SHELL_HISTORY_COUNT (3U)****52.4.6 #define SHELL_HANDLE_SIZE****Value:**

```
(160U + SHELL_HISTORY_COUNT * SHELL_BUFFER_SIZE +
SHELL_BUFFER_SIZE + SERIAL_MANAGER_READ_HANDLE_SIZE + \
SERIAL_MANAGER_WRITE_HANDLE_SIZE)
```

It is the sum of the SHELL_HISTORY_COUNT * SHELL_BUFFER_SIZE + SHELL_BUFFER_SIZE + SERIAL_MANAGER_READ_HANDLE_SIZE + SERIAL_MANAGER_WRITE_HANDLE_SIZE

52.4.7 #define SHELL_USE_COMMON_TASK (0U)**52.4.8 #define SHELL_TASK_PRIORITY (2U)****52.4.9 #define SHELL_TASK_STACK_SIZE (1000U)****52.4.10 #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.
-------------	--------------------------------------

**52.4.11 #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.

52.4.12 #define SHELL_COMMAND(*command*) &g_shellCommand##command

This macro is used to get the shell command pointer. The macro should not be used before the macro SHELL_COMMAND_DEFINE is used.

Parameters

<i>command</i>	The command string of the command. The double quotes do not need. Such as exit for "exit", help for "Help", read for "read".
----------------	--

52.5 Typedef Documentation

52.5.1 typedef shell_status_t(* cmd_function_t)(shell_handle_t shellHandle, int32_t argc, char **argv)

52.6 Enumeration Type Documentation

52.6.1 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.
- kStatus_SHELL_RetUsage* RetUsage for print cmd usage.

52.7 Function Documentation

52.7.1 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 <code>SHELL_HANDLE_SIZE</code> 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: <code>SHELL_HANDLE_DEFINE(shellHandle)</code> ; 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.

52.7.2 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.

52.7.3 **shell_status_t** SHELL_UnregisterCommand (**shell_command_t** * **shellCommand**)

This function is used to unregister the shell command.

Parameters

<i>shellCommand</i>	The command element.
---------------------	----------------------

Return values

<i>kStatus_SHELL_Success</i>	Successfully unregister the command.
------------------------------	--------------------------------------

52.7.4 **shell_status_t** SHELL_Write (**shell_handle_t** **shellHandle**, **const char** * **buffer**, **uint32_t** **length**)

This function is used to send data to the shell output stream.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>buffer</i>	Start address of the data to write.
<i>length</i>	Length of the data to write.

Return values

<i>kStatus_SHELL_Success</i>	Successfully send data.
------------------------------	-------------------------

<i>kStatus_SHELL_Error</i>	An error occurred.
----------------------------	--------------------

52.7.5 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.

52.7.6 shell_status_t SHELL_WriteSynchronization (shell_handle_t *shellHandle*, const char * *buffer*, uint32_t *length*)

This function is used to send data to the shell output stream with OS synchronization, note the function could not be called in ISR.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>buffer</i>	Start address of the data to write.
<i>length</i>	Length of the data to write.

Return values

<i>kStatus_SHELL_Success</i>	Successfully send data.
<i>kStatus_SHELL_Error</i>	An error occurred.

52.7.7 int SHELL_PrintfSynchronization (shell_handle_t *shellHandle*, const char * *formatString*, ...)

Call this function to write a formatted output to the shell output stream with OS synchronization, note the function could not be called in ISR.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>formatString</i>	Format string.

Returns

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

52.7.8 void SHELL_ChangePrompt (shell_handle_t *shellHandle*, char * *prompt*)

Call this function to change shell prompt.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>prompt</i>	The string which will be used for command prompt

Returns

NULL.

52.7.9 void SHELL_PrintPrompt (shell_handle_t *shellHandle*)

Call this function to print shell prompt.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
--------------------	----------------------------------

Returns

NULL.

52.7.10 void SHELL_Task (shell_handle_t *shellHandle*)

The task function for Shell; The function should be polled by upper layer. This function does not return until Shell command exit was called.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
--------------------	----------------------------------

52.7.11 static bool SHELL_checkRunningInIsr (void) [inline], [static]

This function is used to check if code running in ISR.

Return values

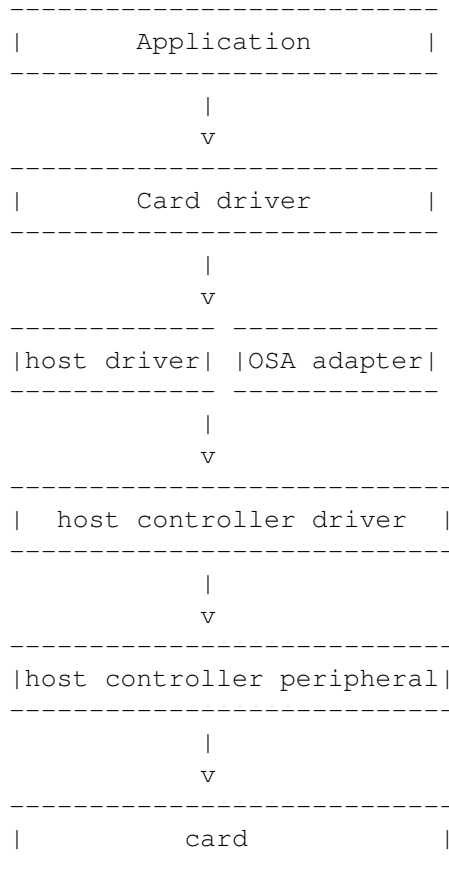
<i>TRUE</i>	if code running in ISR.
-------------	-------------------------

Chapter 53

Cards: Secure Digital Card/Embedded MultiMedia Card/SD-IO Card

53.1 Overview

The MCUXpresso SDK provides drivers to access the Secure Digital Card(up to v3.0), Embedded Multi-Media Card(up to v5.0) and sdio card(up to v3.0) based on the SDHC/USDHC/SDIF driver. Here is a simple block diagram about the drivers:



Modules

- [MMC Card Driver](#)
- [SD Card Driver](#)
- [SDIO Card Driver](#)
- [SDMMC Common](#)
- [SDMMC HOST Driver](#)
- [SDMMC OSA](#)

53.2 SDIO Card Driver

53.2.1 Overview

The SDIO card driver provide card initialization/IO direct and extend command interface.

53.2.2 SDIO CARD Operation

error log support

Not supported yet.

User configurable

Board dependency

Mutual exclusive access support for RTOS

SDIO driver has added mutual exclusive access support for init/deinit/write/read/erase function. Please note that the card init function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
SDIO_Deinit(card); /* This function will destroy the created mutex */
SDIO_Init(card);
```

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, 4U, 1U)`) /*2.4.1*/
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_HostDoReset](#) (sdio_card_t *card)
reset the host.
- 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_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.

53.2.3 Data Structure Documentation

53.2.3.1 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`
used to record current enabled io interrupt index
- `uint8_t ioIntNums`
used to record total enabled io interrupt numbers
- `sdmhc_osa_mutex_t lock`
card access lock

Field Documentation

(1) `bool sdio_card_t::noInternalAlign`

If disable, sdmhc will not make sure the data buffer address is word align, otherwise all the transfer are align to low level driver

53.2.4 Macro Definition Documentation

53.2.4.1 `#define FSL_SDIO_DRIVER_VERSION (MAKE_VERSION(2U, 4U, 1U)) /*2.4.1*/`

53.2.5 Enumeration Type Documentation

53.2.5.1 `enum sdio_io_direction_t`

Enumerator

kSDIO_IORead io read
kSDIO_IOWrite io write

53.2.6 Function Documentation

53.2.6.1 `status_t SDIO_Init (sdio_card_t * card)`

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
* SDIO_Deinit (card);
* SDIO_Init (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>	

53.2.6.2 void SDIO_Deinit (sdio_card_t * card)

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

53.2.6.3 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*.

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization


```
* SDIO_CardDeinit (card);
* SDIO_CardInit (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_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.

53.2.6.4 void SDIO_CardDeinit (sdio_card_t * card)

This function deinitializes the specific card.

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

53.2.6.5 status_t SDIO_HostInit (sdio_card_t * card)

This function deinitializes the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

53.2.6.6 void SDIO_HostDeinit (sdio_card_t * card)

This function deinitializes the host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

53.2.6.7 void SDIO_HostDoReset (sdio_card_t * card)

This function reset the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

53.2.6.8 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.

53.2.6.9 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>	

53.2.6.10 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>	

53.2.6.11 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>	

53.2.6.12 **status_t SDIO_CardReset (sdio_card_t * card)**

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_Success</i>	

53.2.6.13 **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>	

53.2.6.14 `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>	

53.2.6.15 `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>	

53.2.6.16 `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.

53.2.6.17 bool SDIO_IsCardPresent (sdio_card_t * *card*)

Parameters

<i>card</i>	card descriptor.
-------------	------------------

53.2.6.18 status_t SDIO_IO_Write_Direct (sdio_card_t * *card*, sdio_func_num_t *func*, uint32_t *regAddr*, uint8_t * *data*, bool *raw*)

Please note it is a thread safe function.

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_</i> <i>TransferFailed</i>	
<i>kStatus_Success</i>	

53.2.6.19 status_t SDIO_IO_Read_Direct (sdio_card_t * *card*, sdio_func_num_t *func*, uint32_t *regAddr*, uint8_t * *data*)

Please note it is a thread safe function.

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>	

53.2.6.20 `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)`

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>direction</i>	io access direction, please reference <code>sdio_io_direction_t</code> .
<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, <code>dataOut</code> should not be NULL.

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_Success</i>	

53.2.6.21 `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)`

Please note it is a thread safe function.

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>	

53.2.6.22 `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)`

Please note it is a thread safe function.

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>	

53.2.6.23 `status_t SDIO_EnableInterrupt (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>	

53.2.6.24 **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>	

53.2.6.25 **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>	

53.2.6.26 `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>	

53.2.6.27 **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>	

53.2.6.28 **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>	
------------------------	--

53.2.6.29 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_</i> <i>TransferFailed</i>	
<i>kStatus_Success</i>	

53.2.6.30 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 note it is a thread safe function.

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.

53.2.6.31 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.

53.2.6.32 status_t SDIO_HandlePendingIOInterrupt (sdio_card_t * card)

This function is used to handle the pending io interrupt. To reigster a IO IRQ handler,

```
* 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>	

53.3 SD Card Driver

53.3.1 Overview

The SDCARD driver provide card initialization/read/write/erase interface.

53.3.2 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
{
    sdmmc_host_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;
    sdmmc_osa_mutex_t lock;
} 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_USDHClClockConfiguration();

/* 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.

1. bool noInternalAlign

Sdmmc include an address align internal buffer(to use host controller internal DMA), to improve read/write performance while application cannot make sure the data address used to read/write is align, set it to true will achieve a better performance.

2. sd_timing_mode_t currentTiming

It is used to indicate the currentTiming the card is working on, however sdmmc also support preset timing mode, then sdmmc will try to switch to this timing first, if failed, a valid timing will switch to automatically. Generally, user may not set this variable if you don't know what kind of timing the card support, sdmmc will switch to the highest timing which the card support.

3. sd_driver_strength_t driverStrength

Choose a valid card driver strength if application required and call SD_SetDriverStrength in application.

4. sd_max_current_t maxCurrent

Choose a valid card current if application required and call SD_SetMaxCurrent in application.

Mutual exclusive access support for RTOS

SDCARD driver has added mutual exclusive access support for init/deinit/write/read/erase function. Please note that the card init function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```

SD_Deinit(card); /* This function will destroy the created mutex */
SD_Init(card);

```

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, 4U, 2U)`) /*2.4.2*/
Driver version.

Enumerations

- enum {
`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_SetCardPower` (`sd_card_t *card`, `bool enable`)
set card power.
- `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 block-Count`)

- *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
- `status_t SD_PollingCardStatusBusy (sd_card_t *card, uint32_t timeoutMs)`
Polling card idle status.

53.3.3 Data Structure Documentation

53.3.3.1 struct sd_card_t

Define the card structure including the necessary fields to identify and describe the card.

Data Fields

- `sdtmmchost_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 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
- sdmmc_osa_mutex_t **lock**
card access lock

53.3.4 Macro Definition Documentation

53.3.4.1 #define FSL_SD_DRIVER_VERSION (MAKE_VERSION(2U, 4U, 2U)) /*2.4.2*/

53.3.5 Enumeration Type Documentation

53.3.5.1 anonymous enum

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

53.3.6 Function Documentation

53.3.6.1 status_t SD_Init (sd_card_t * card)

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.

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

* `SD_Deinit(card);`

```
* SD_Init(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_Not-SupportYet</i>	Card not support.
<i>kStatus_SDMMC_Hand-ShakeOperation-ConditionFailed</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_Set-DataBusWidthFailed</i>	Set bus width failed.
<i>kStatus_SDMMC_Switch-BusTimingFailed</i>	Switch high speed failed.
<i>kStatus_SDMMC_Set-CardBlockSizeFailed</i>	Set card block size failed.

<i>kStatus_Success</i>	Operate successfully.
------------------------	-----------------------

53.3.6.2 void SD_Deinit (sd_card_t * card)

This function deinitializes the specific card and host. Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

53.3.6.3 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*.

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
* SD_CardDeinit (card);
* SD_CardInit (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_Not-SupportYet</i>	Card not support.
<i>kStatus_SDMMC_Hand-ShakeOperation-ConditionFailed</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_Set-DataBusWidthFailed</i>	Set bus width failed.
<i>kStatus_SDMMC_Switch-BusTimingFailed</i>	Switch high speed failed.
<i>kStatus_SDMMC_Set-CardBlockSizeFailed</i>	Set card block size failed.
<i>kStatus_Success</i>	Operate successfully.

53.3.6.4 void SD_CardDeinit (sd_card_t * card)

This function deinitializes the specific card. Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

53.3.6.5 status_t SD_HostInit (sd_card_t * card)

This function deinitializes the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

53.3.6.6 void SD_HostDeinit (sd_card_t * card)

This function deinitializes the host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

53.3.6.7 void SD_HostDoReset (sd_card_t * *card*)

This function reset the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

53.3.6.8 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.

53.3.6.9 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.

53.3.6.10 bool SD_IsCardPresent (sd_card_t * *card*)

Parameters

<i>card</i>	card descriptor.
-------------	------------------

53.3.6.11 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.

53.3.6.12 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.

53.3.6.13 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_Send-ApplicationCommand-Failed</i>	send application command failed.

<i>kStatus_Success</i>	Operate successfully.
------------------------	-----------------------

53.3.6.14 **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.

Please note it is a thread safe function.

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_Card-NotSupport</i>	Card not support.
<i>kStatus_SDMMC_Not-SupportYet</i>	Not support now.
<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.

53.3.6.15 **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.

Please note,

1. It is a thread safe function.
2. It is a async write function which means that the card status may still busy after the function return.

Application can call function `SD_PollingCardStatusBusy` to wait card status idle after the write operation.

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.

53.3.6.16 `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.

Please note,

1. It is a thread safe function.
2. It is a async erase function which means that the card status may still busy after the function return. Application can call function `SD_PollingCardStatusBusy` to wait card status idle after the erase operation.

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.

53.3.6.17 **status_t SD_SetDriverStrength (sd_card_t * *card*, sd_driver_strength_t *driverStrength*)**

Parameters

<i>card</i>	Card descriptor.
<i>driverStrength</i>	Driver strength

53.3.6.18 **status_t SD_SetMaxCurrent (sd_card_t * *card*, sd_max_current_t *maxCurrent*)**

Parameters

<i>card</i>	Card descriptor.
<i>maxCurrent</i>	Max current

53.3.6.19 **status_t SD_PollingCardStatusBusy (sd_card_t * *card*, uint32_t *timeoutMs*)**

This function can be used to polling the status from busy to Idle, the function will return if the card status idle or timeout.

Parameters

<i>card</i>	Card descriptor.
<i>timeoutMs</i>	polling card status timeout value.

Return values

<i>kStatus_Success</i>	Operate successfully.
<i>kStatus_SDMMC_Wait-WriteCompleteFailed</i>	CMD13 transfer failed.
<i>kStatus_SDMMC_-PollingCardIdle-Failed,polling</i>	card DAT0 idle failed.

53.4 MMC Card Driver

53.4.1 Overview

The MMCCARD driver provide card initialization/read/write/erase interface.

53.4.2 MMC CARD Operation

error log support

Not support yet

User configurable

Board dependency

Mutual exclusive access support for RTOS

MMCCARD driver has added mutual exclusive access support for init/deinit/write/read/erase function. Please note that the card init function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization.

```
MMC_Deinit(card); /* This function will destroy the created mutex */
MMC_Init(card);
```

Typical use case

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

Data Structures

- struct `mmc_usr_param_t`
card user parameter [More...](#)
- struct `mmc_card_t`
mmc card state [More...](#)

Macros

- #define `FSL_MMC_DRIVER_VERSION` (`MAKE_VERSION(2U, 5U, 0U)`) /*2.5.0*/
Middleware mmc version.

Typedefs

- typedef void(* [mmc_io_strength_t](#))(uint32_t busFreq)
card io strength control

Enumerations

- enum {
[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.
- enum [mmc_sleep_awake_t](#) {
[kMMC_Sleep](#) = 1U,
[kMMC_Awake](#) = 0U }
mmccard sleep/awake state

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)
Initializes 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_HostDoReset](#) ([mmc_card_t](#) *card)
Resets the host.
- void [MMC_HostReset](#) (SDMMCHOST_CONFIG *host)
Resets the host.
- void [MMC_SetCardPower](#) ([mmc_card_t](#) *card, bool enable)

- *Sets 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)`
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 continuous 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 bootPartitionWP)`
MMC card set boot partition write protect.
- `status_t MMC_EnableCacheControl (mmc_card_t *card, bool enable)`
MMC card cache control function.
- `status_t MMC_FlushCache (mmc_card_t *card)`
MMC card cache flush function.
- `status_t MMC_SetSleepAwake (mmc_card_t *card, mmc_sleep_awake_t state)`
MMC sets card sleep awake state.
- `status_t MMC_PollingCardStatusBusy (mmc_card_t *card, bool checkStatus, uint32_t timeoutMs)`
Polling card idle status.

53.4.3 Data Structure Documentation

53.4.3.1 struct mmc_usr_param_t

Data Fields

- `mmc_io_strength_t ioStrength`
switch sd io strength
- `uint32_t maxFreq`
board support maximum frequency
- `uint32_t capability`
board capability flag

53.4.3.2 struct mmc_card_t

Defines the card structure including the necessary fields to identify and describe the card.

Data Fields

- `sdmchost_t * host`
Host information.
- `mmc_usr_param_t usrParam`
user parameter
- `bool isHostReady`
Use this flag to indicate if host re-init needed 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`
indicates the current work timing mode
- `mmc_data_bus_width_t busWidth`
indicates the current work bus width

- `sdmmc_osa_mutex_t lock`
card access lock

Field Documentation

(1) `bool mmc_card_t::noInterAlAlign`

If disabled, `sdmmc` will not make sure the data buffer address is word align, otherwise all the transfer are aligned to low level driver.

53.4.4 Macro Definition Documentation

53.4.4.1 `#define FSL_MMC_DRIVER_VERSION (MAKE_VERSION(2U, 5U, 0U)) /*2.5.0*/`

53.4.5 Enumeration Type Documentation

53.4.5.1 anonymous enum

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

53.4.5.2 `enum mmc_sleep_aware_t`

Enumerator

kMMC_Sleep MMC card sleep.
kMMC_Awake MMC card awake.

53.4.6 Function Documentation

53.4.6.1 `status_t MMC_Init (mmc_card_t * card)`

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex to be created redundantly, application must follow bellow sequence for card re-initialization:

```

MMC_Deinit(card);
MMC_Init(card);

```

*

Return values

<i>kStatus_SDMMC_Host-NotReady</i>	Host is not ready.
<i>kStatus_SDMMC_Go-IdleFailed</i>	Going idle failed.
<i>kStatus_SDMMC_Hand-ShakeOperation-ConditionFailed</i>	Sending operation condition failed.
<i>kStatus_SDMMC_All-SendCidFailed</i>	Sending CID failed.
<i>kStatus_SDMMC_Set-RelativeAddressFailed</i>	Setgng relative address failed.
<i>kStatus_SDMMC_Send-CsdFailed</i>	Sending CSD failed.
<i>kStatus_SDMMC_Card-NotSupport</i>	Card not support.
<i>kStatus_SDMMC_Select-CardFailed</i>	Sending SELECT_CARD command failed.
<i>kStatus_SDMMC_Send-ExtendedCsdFailed</i>	Sending EXT_CSD failed.
<i>kStatus_SDMMC_Set-DataBusWidthFailed</i>	Setting bus width failed.

<i>kStatus_SDMMC_Switch-BusTimingFailed</i>	Switching high speed failed.
<i>kStatus_SDMMC_Set-CardBlockSizeFailed</i>	Setting card block size failed.
<i>kStatus_SDMMC_Set-PowerClassFail</i>	Setting card power class failed.
<i>kStatus_Success</i>	Operation succeeded.

53.4.6.2 void MMC_Deinit (mmc_card_t * card)

Note

It is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

53.4.6.3 status_t MMC_CardInit (mmc_card_t * card)

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex to be created redundantly, application must follow bellow sequence for card re-initialization:

```

MMC_CardDeinit (card);
MMC_CardInit (card);
*
```

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_Host-NotReady</i>	Host is not ready.
------------------------------------	--------------------

<i>kStatus_SDMMC_GoIdleFailed</i>	Going idle failed.
<i>kStatus_SDMMC_HandShakeOperationConditionFailed</i>	Sending operation condition failed.
<i>kStatus_SDMMC_AllSendCidFailed</i>	Sending CID failed.
<i>kStatus_SDMMC_SetRelativeAddressFailed</i>	Setting relative address failed.
<i>kStatus_SDMMC_SendCsdFailed</i>	Sending CSD failed.
<i>kStatus_SDMMC_CardNotSupport</i>	Card not support.
<i>kStatus_SDMMC_SelectCardFailed</i>	Sending SELECT_CARD command failed.
<i>kStatus_SDMMC_SendExtendedCsdFailed</i>	Sending EXT_CSD failed.
<i>kStatus_SDMMC_SetDataBusWidthFailed</i>	Setting bus width failed.
<i>kStatus_SDMMC_SwitchBusTimingFailed</i>	Switching high speed failed.
<i>kStatus_SDMMC_SetCardBlockSizeFailed</i>	Setting card block size failed.
<i>kStatus_SDMMC_SetPowerClassFail</i>	Setting card power class failed.
<i>kStatus_Success</i>	Operation succeeded.

53.4.6.4 void MMC_CardDeinit (mmc_card_t * card)

Note

It is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

53.4.6.5 `status_t MMC_HostInit (mmc_card_t * card)`

This function deinitializes the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

53.4.6.6 `void MMC_HostDeinit (mmc_card_t * card)`

This function deinitializes the host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

53.4.6.7 `void MMC_HostDoReset (mmc_card_t * card)`

This function resets the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

53.4.6.8 `void MMC_HostReset (SDMMCHOST_CONFIG * host)`

Deprecated Do not use this function. It has been superseded by [MMC_HostDoReset](#). This function resets the specific host.

Parameters

<i>host</i>	Host descriptor.
-------------	------------------

53.4.6.9 `void MMC_SetCardPower (mmc_card_t * card, bool enable)`

Parameters

<i>card</i>	Card descriptor.
<i>enable</i>	True is powering on, false is powering off.

53.4.6.10 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.

53.4.6.11 status_t MMC_ReadBlocks (mmc_card_t * *card*, uint8_t * *buffer*, uint32_t *startBlock*, uint32_t *blockCount*)

Note

It is a thread safe function.

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_SetBlockCountFailed</i>	Setting block count failed.
<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_StopTransmissionFailed</i>	Stopping transmission failed.
<i>kStatus_Success</i>	Operation succeeded.

53.4.6.12 `status_t MMC_WriteBlocks (mmc_card_t * card, const uint8_t * buffer, uint32_t startBlock, uint32_t blockCount)`

Note

1. It is a thread safe function.
2. It is an async write function which means that the card status may still be busy after the function returns. Application can call function `MMC_PollingCardStatusBusy` to wait for the card status to be idle after the write operation.

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_NotSupportYet</i>	Not support now.
<i>kStatus_SDMMC_SetBlockCountFailed</i>	Setting block count failed.
<i>kStatus_SDMMC_WaitWriteCompleteFailed</i>	Sending status failed.

<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_StopTransmissionFailed</i>	Stop transmission failed.
<i>kStatus_Success</i>	Operation succeeded.

53.4.6.13 `status_t MMC_EraseGroups (mmc_card_t * card, uint32_t startGroup, uint32_t endGroup)`

The erase command is best used to erase the entire device or a partition. Erase group is the smallest erase unit in MMC card. The erase range is [startGroup, endGroup].

Note

1. It is a thread safe function.
2. This function always polls card busy status according to the timeout value defined in the card register after all the erase command sent out.

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>	Operation succeeded.

53.4.6.14 `status_t MMC_SelectPartition (mmc_card_t * card, mmc_access_partition_t partitionNumber)`

Note

It is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>partition-Number</i>	The partition number.

Return values

<i>kStatus_SDMMC_-ConfigureExtendedCsd-Failed</i>	Configuring EXT_CSD failed.
<i>kStatus_Success</i>	Operation succeeded.

53.4.6.15 `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>	Configuring EXT_CSD failed.
<i>kStatus_SDMMC_-ConfigureBootFailed</i>	Configuring boot failed.
<i>kStatus_Success</i>	Operation succeeded.

53.4.6.16 `status_t MMC_StartBoot (mmc_card_t * card, const mmc_boot_config_t * mmcConfig, uint8_t * buffer, sdmmchost_boot_config_t * hostConfig)`

Parameters

<i>card</i>	Card descriptor.
<i>mmcConfig</i>	The mmc Boot configuration structure.
<i>buffer</i>	Address to receive data.
<i>hostConfig</i>	Host boot configurations.

Return values

<i>kStatus_Fail</i>	Failed.
<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_GoIdleFailed</i>	Resetting card failed.
<i>kStatus_Success</i>	Operation succeeded.

53.4.6.17 status_t MMC_SetBootConfigWP (mmc_card_t * card, uint8_t wp)

Parameters

<i>card</i>	Card descriptor.
<i>wp</i>	Write protect value.

53.4.6.18 status_t MMC_ReadBootData (mmc_card_t * card, uint8_t * buffer, sdmmchost_boot_config_t * hostConfig)

Parameters

<i>card</i>	Card descriptor.
<i>buffer</i>	Buffer address.
<i>hostConfig</i>	Host boot configurations.

53.4.6.19 status_t MMC_StopBoot (mmc_card_t * card, uint32_t bootMode)

Parameters

<i>card</i>	Card descriptor.
<i>bootMode</i>	Boot mode.

53.4.6.20 `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.

53.4.6.21 `status_t MMC_EnableCacheControl (mmc_card_t * card, bool enable)`

The mmc device's cache is enabled by the driver by default. The cache should in typical case reduce the access time (compared to an access to the main nonvolatile storage) for both write and read.

Parameters

<i>card</i>	Card descriptor.
<i>enable</i>	True is enabling the cache, false is disabling the cache.

53.4.6.22 `status_t MMC_FlushCache (mmc_card_t * card)`

A Flush operation refers to the requirement, from the host to the device, to write the cached data to the nonvolatile memory. Prior to a flush, the device may autonomously write data to the nonvolatile memory, but after the flush operation all data in the volatile area must be written to nonvolatile memory. There is no requirement for flush due to switching between the partitions. (Note: This also implies that the cache data shall not be lost when switching between partitions). Cached data may be lost in SLEEP state, so host should flush the cache before placing the device into SLEEP state.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

53.4.6.23 `status_t MMC_SetSleepAwake (mmc_card_t * card, mmc_sleep_awake_t state)`

The Sleep/Awake command is used to initiate the state transition between Standby state and Sleep state. The memory device indicates the transition phase busy by pulling down the DAT0 line. The Sleep/Standby state is reached when the memory device stops pulling down the DAT0 line, then the function returns.

Parameters

<i>card</i>	Card descriptor.
<i>state</i>	The sleep/awake command argument, refer to mmc_sleep_awake_t .

Return values

<i>kStatus_SDMMC_Not-SupportYet</i>	Indicates the memory device doesn't support the Sleep/Awake command.
<i>kStatus_SDMMC_-TransferFailed</i>	Indicates command transferred fail.
<i>kStatus_SDMMC_-PollingCardIdleFailed</i>	Indicates polling DAT0 busy timeout.
<i>kStatus_SDMMC_-DeselectCardFailed</i>	Indicates deselect card command failed.
<i>kStatus_SDMMC_Select-CardFailed</i>	Indicates select card command failed.
<i>kStatus_Success</i>	Indicates the card state switched successfully.

53.4.6.24 `status_t MMC_PollingCardStatusBusy (mmc_card_t * card, bool checkStatus, uint32_t timeoutMs)`

This function can be used to poll the status from busy to idle, the function will return with the card status being idle or timeout or command failed.

Parameters

<i>card</i>	Card descriptor.
<i>checkStatus</i>	True is send CMD and read DAT0 status to check card status, false is read DAT0 status only.
<i>timeoutMs</i>	Polling card status timeout value.

Return values

<i>kStatus_SDMMC_Card- StatusIdle</i>	Card is idle.
<i>kStatus_SDMMC_Card- StatusBusy</i>	Card is busy.
<i>kStatus_SDMMC_- TransferFailed</i>	Command transfer failed.
<i>kStatus_SDMMC_Switch- Failed</i>	Status command reports switch error.

53.5 SDMMC HOST Driver

53.5.1 Overview

The host adapter driver provide adapter for blocking/non_blocking mode.

Modules

- [USDHC HOST adapter Driver](#)

53.6 SDMMC OSA

53.6.1 Overview

The sdmmc osa adapter provide interface of os adapter.

Data Structures

- struct [sdmmc_osa_event_t](#)
sdmmc osa event More...
- struct [sdmmc_osa_mutex_t](#)
sdmmc osa mutex More...

Macros

- #define [SDMMC_OSA_EVENT_TRANSFER_CMD_SUCCESS](#) (1UL << 0U)
transfer event
- #define [SDMMC_OSA_EVENT_CARD_INSERTED](#) (1UL << 8U)
card detect event, start from index 8
- #define [SDMMC_OSA_POLLING_EVENT_BY_SEMPHORE](#) 1
enable semaphore by default

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 eventType, 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.
- [status_t SDMMC_OSAMutexCreate](#) (void *mutexHandle)
Create a mutex.
- [status_t SDMMC_OSAMutexLock](#) (void *mutexHandle, uint32_t millisec)
set event.
- [status_t SDMMC_OSAMutexUnlock](#) (void *mutexHandle)
Get event flag.
- [status_t SDMMC_OSAMutexDestroy](#) (void *mutexHandle)
Delete mutex.

- void [SDMMC_OSADelay](#) (uint32_t milliseconds)
sdmmc delay.
- uint32_t [SDMMC_OSADelayUs](#) (uint32_t microseconds)
sdmmc delay us.

53.6.2 Data Structure Documentation

53.6.2.1 struct sdmmc_osa_event_t

53.6.2.2 struct sdmmc_osa_mutex_t

53.6.3 Function Documentation

53.6.3.1 status_t SDMMC_OSAEventCreate (void * *eventHandle*)

Parameters

<i>eventHandle</i>	event handle.
--------------------	---------------

Return values

<i>kStatus_Fail</i>	or kStatus_Success.
---------------------	---------------------

53.6.3.2 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.
---------------------	---------------------

53.6.3.3 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.
---------------------	---------------------

53.6.3.4 status_t SDMMC_OSAEventGet (void * eventHandle, uint32_t eventType, uint32_t * flag)

Parameters

<i>eventHandle</i>	event handle.
<i>eventType</i>	event type.
<i>flag</i>	pointer to store event value.

Return values

<i>kStatus_Fail</i>	or kStatus_Success.
---------------------	---------------------

53.6.3.5 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.
---------------------	---------------------

53.6.3.6 status_t SDMMC_OSAEventDestroy (void * *eventHandle*)

Parameters

<i>eventHandle</i>	The event handle.
--------------------	-------------------

53.6.3.7 status_t SDMMC_OSAMutexCreate (void * *mutexHandle*)

Parameters

<i>mutexHandle</i>	mutex handle.
--------------------	---------------

Return values

<i>kStatus_Fail</i>	or kStatus_Success.
---------------------	---------------------

53.6.3.8 status_t SDMMC_OSAMutexLock (void * *mutexHandle*, uint32_t *millisec*)

Parameters

<i>mutexHandle</i>	mutex handle.
--------------------	---------------

<i>millisec</i>	The maximum number of milliseconds to wait for the mutex. If the mutex is locked, Pass the value <code>osaWaitForever_c</code> will wait indefinitely, pass 0 will return <code>KOSA_StatusTimeout</code> immediately.
-----------------	--

Return values

<i>kStatus_Fail</i>	or kStatus_Success.
---------------------	---------------------

53.6.3.9 status_t SDMMC_OSAMutexUnlock (void * *mutexHandle*)

Parameters

<i>mutexHandle</i>	mutex handle.
--------------------	---------------

Return values

<i>kStatus_Fail</i>	or kStatus_Success.
---------------------	---------------------

53.6.3.10 status_t SDMMC_OSAMutexDestroy (void * *mutexHandle*)

Parameters

<i>mutexHandle</i>	The mutex handle.
--------------------	-------------------

53.6.3.11 void SDMMC_OSADelay (uint32_t *milliseconds*)

Parameters

<i>milliseconds</i>	time to delay
---------------------	---------------

53.6.3.12 uint32_t SDMMC_OSADelayUs (uint32_t *microseconds*)

Parameters

<i>microseconds</i>	time to delay
---------------------	---------------

Returns

actual delayed microseconds

53.6.4 USDHC HOST adapter Driver

53.6.4.1 Overview

The USDHC host adapter driver provide adapter for blocking/non_blocking mode.

Cache maintain capability

To maintain data integrity during DMA operations on the platform that has cache, host driver provide a cache maintain functionality by set: `host.enableCacheControl = kSDMMCHOST_CacheControlRW-Buffer` This is used for only when the low level driver cache maintain functionality is disabled. It is suggest that the address of buffer used for read/write is align with cache line size.

Cache line alignment maintain capability

when application submit a transfer request that the data buffer address is not align with cache line size, the potential data loss may happen during driver maintain the data cache, to avoid such issue happens, `sdmmc usdhc` host driver provides support on convert the unalign data transfer into align data transfer, if application would like to use the feature, please enable this functionality by define below macro firstly, `#define SDMMCHOST_ENABLE_CACHE_LINE_ALIGN_TRANSFER 1` And then call `SDMMCHOST_InstallCacheAlignBuffer` to install a cache line size align buffer, please note the installed buffer size must not small than $2 * \text{cache line size}$. Please note that this functionality is support by the non blocking adapter only.

Data Structures

- struct `sdmmchost_t`
sdmmc host handler [More...](#)

Macros

- `#define FSL_SDMMC_HOST_ADAPTER_VERSION (MAKE_VERSION(2U, 6U, 3U)) /*2.6.3*/`
Middleware adapter version.
- `#define SDMMCHOST_SUPPORT_HIGH_SPEED (1U)`
sdmmc host misc capability
- `#define SDMMCHOST_SUPPORT_DDR50 (SDMMCHOST_SUPPORT_DDR_MODE)`
sdmmc host sdcard DDR50 mode capability
- `#define SDMMCHOST_SUPPORT_SDR104 (1U)`
sdmmc host sdcard SDR50 mode capability
- `#define SDMMCHOST_SUPPORT_SDR50 (1U)`
sdmmc host sdcard SDR104/mmccard HS200 mode capability
- `#define SDMMCHOST_SUPPORT_HS400 (1U)`
sdmmc host mmccard HS400 mode capability

- #define `SDMMCHOST_INSTANCE_SUPPORT_8_BIT_WIDTH`(host) FSL_FEATURE_USDHC_INSTANCE_SUPPORT_8_BIT_WIDTHn(host->hostController.base)
sdmmc host instance capability
- #define `SDMMCHOST_DATA3_DETECT_CARD_DELAY` (10U)
sdmmchost delay for DAT3 detect card
- #define `SDMMCHOST_DMA_DESCRIPTOR_BUFFER_ALIGN_SIZE` (4U)
SDMMC host dma descriptor buffer address align size.
- #define `SDMMCHOST_STANDARD_TUNING_START` (10U)
tuning configuration
- #define `SDMMCHOST_TUINIG_STEP` (2U)
standard tuning stBep

Typedefs

- typedef `usdhc_transfer_t` `sdmmchost_transfer_t`
sdmmc host transfer function

Enumerations

- enum {
`kSDMMCHOST_SupportHighSpeed` = 1U << 0U,
`kSDMMCHOST_SupportSuspendResume` = 1U << 1U,
`kSDMMCHOST_SupportVoltage3v3` = 1U << 2U,
`kSDMMCHOST_SupportVoltage3v0` = 1U << 3U,
`kSDMMCHOST_SupportVoltage1v8` = 1U << 4U,
`kSDMMCHOST_SupportVoltage1v2` = 1U << 5U,
`kSDMMCHOST_Support4BitDataWidth` = 1U << 6U,
`kSDMMCHOST_Support8BitDataWidth` = 1U << 7U,
`kSDMMCHOST_SupportDDRMMode` = 1U << 8U,
`kSDMMCHOST_SupportDetectCardByData3` = 1U << 9U,
`kSDMMCHOST_SupportDetectCardByCD` = 1U << 10U,
`kSDMMCHOST_SupportAutoCmd12` = 1U << 11U,
`kSDMMCHOST_SupportSDR104` = 1U << 12U,
`kSDMMCHOST_SupportSDR50` = 1U << 13U,
`kSDMMCHOST_SupportHS200` = 1U << 14U,
`kSDMMCHOST_SupportHS400` = 1U << 15U }
sdmmc host capability
- enum {
`kSDMMCHOST_EndianModeBig` = 0U,
`kSDMMCHOST_EndianModeHalfWordBig` = 1U,
`kSDMMCHOST_EndianModeLittle` = 2U }
host Endian mode corresponding to driver define
- enum {
`kSDMMCHOST_StandardTuning` = 0U,
`kSDMMCHOST_ManualTuning` = 1U }
sdmmc host tuning type

- enum {
 kSDMMCHOST_NoCacheControl = 0U,
 kSDMMCHOST_CacheControlRWBuffer = 1U }
 sdmmc host maintain cache flag

USDHC 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.
- static uint32_t **SDMMCHOST_GetSignalLineStatus** (sdmmchost_t *host, uint32_t signalLine)
 Get signal line status.
- static void **SDMMCHOST_EnableCardInt** (sdmmchost_t *host, bool enable)
 enable card interrupt.
- static void **SDMMCHOST_EnableDDRMode** (sdmmchost_t *host, bool enable, uint32_t nibble-Pos)
 enable DDR mode.
- static void **SDMMCHOST_EnableHS400Mode** (sdmmchost_t *host, bool enable)
 enable HS400 mode.
- static void **SDMMCHOST_EnableStrobeDll** (sdmmchost_t *host, bool enable)
 enable STROBE DLL.
- **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 *host-Config, uint8_t *buffer)
 read boot data.
- static void **SDMMCHOST_EnableBoot** (sdmmchost_t *host, bool enable)
 enable boot mode.
- **status_t SDMMCHOST_CardIntInit** (sdmmchost_t *host, void *sdioInt)
 card interrupt function.
- static void **SDMMCHOST_ForceClockOn** (sdmmchost_t *host, bool enable)
 force card clock on.
- void **SDMMCHOST_SwitchToVoltage** (sdmmchost_t *host, uint32_t voltage)
 switch to voltage.
- **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 [SDMMCCHOST_SetCardPower](#) ([sdmmchost_t](#) *host, bool enable)
host power off card function.
- [status_t SDMMCCHOST_TransferFunction](#) ([sdmmchost_t](#) *host, [sdmmchost_transfer_t](#) *content)
host transfer function.
- [status_t SDMMCCHOST_ExecuteTuning](#) ([sdmmchost_t](#) *host, [uint32_t](#) tuningCmd, [uint32_t](#) *revBuf, [uint32_t](#) blockSize)
sdmmc host excute tuning.
- void [SDMMCCHOST_Reset](#) ([sdmmchost_t](#) *host)
host reset function.
- void [SDMMCCHOST_ConvertDataToLittleEndian](#) ([sdmmchost_t](#) *host, [uint32_t](#) *data, [uint32_t](#) wordSize, [uint32_t](#) format)
sdmmc host convert data sequence to little endian sequence

53.6.4.2 Data Structure Documentation

53.6.4.2.1 struct [sdmmchost_t](#)

Data Fields

- [usdhc_host_t](#) [hostController](#)
host configuration
- void * [dmaDesBuffer](#)
DMA descriptor buffer address.
- [uint32_t](#) [dmaDesBufferWordsNum](#)
DMA descriptor buffer size in byte.
- [usdhc_handle_t](#) [handle](#)
host controller handler
- [uint32_t](#) [capability](#)
host controller capability
- [uint32_t](#) [maxBlockCount](#)
host controller maximum block count
- [uint32_t](#) [maxBlockSize](#)
host controller maximum block size
- [uint8_t](#) [tuningType](#)
host tuning type
- [sdmmc_osa_event_t](#) [hostEvent](#)
host event handler
- void * [cd](#)
card detect
- void * [cardInt](#)
call back function for card interrupt
- [sdmmc_osa_mutex_t](#) [lock](#)
host access lock

53.6.4.3 Macro Definition Documentation

53.6.4.3.1 #define FSL_SDMMC_HOST_ADAPTER_VERSION (MAKE_VERSION(2U, 6U, 3U))
*/*2.6.3*/*

53.6.4.3.2 #define SDMMCHOST_STANDARD_TUNING_START (10U)

standard tuning start point

53.6.4.4 Enumeration Type Documentation

53.6.4.4.1 anonymous enum

Enumerator

kSDMMCHOST_SupportHighSpeed high speed capability
kSDMMCHOST_SupportSuspendResume suspend resume capability
kSDMMCHOST_SupportVoltage3v3 3V3 capability
kSDMMCHOST_SupportVoltage3v0 3V0 capability
kSDMMCHOST_SupportVoltage1v8 1V8 capability
kSDMMCHOST_SupportVoltage1v2 1V2 capability
kSDMMCHOST_Support4BitDataWidth 4 bit data width capability
kSDMMCHOST_Support8BitDataWidth 8 bit data width capability
kSDMMCHOST_SupportDDRMode DDR mode capability.
kSDMMCHOST_SupportDetectCardByData3 data3 detect card capability
kSDMMCHOST_SupportDetectCardByCD CD detect card capability.
kSDMMCHOST_SupportAutoCmd12 auto command 12 capability
kSDMMCHOST_SupportSDR104 SDR104 capability.
kSDMMCHOST_SupportSDR50 SDR50 capability.
kSDMMCHOST_SupportHS200 HS200 capability.
kSDMMCHOST_SupportHS400 HS400 capability.

53.6.4.4.2 anonymous enum

Enumerator

kSDMMCHOST_EndianModeBig Big endian mode.
kSDMMCHOST_EndianModeHalfWordBig Half word big endian mode.
kSDMMCHOST_EndianModeLittle Little endian mode.

53.6.4.4.3 anonymous enum

Enumerator

kSDMMCHOST_StandardTuning standard tuning type
kSDMMCHOST_ManualTuning manual tuning type

53.6.4.4.4 anonymous enum

Enumerator

kSDMMCHOST_NoCacheControl sdmmc host cache control disabled
kSDMMCHOST_CacheControlRWBuffer sdmmc host cache control read/write buffer

53.6.4.5 Function Documentation

53.6.4.5.1 void SDMMCHOST_SetCardBusWidth (sdmmchost_t * *host*, uint32_t *dataBusWidth*)

Parameters

<i>host</i>	host handler
<i>dataBusWidth</i>	data bus width

53.6.4.5.2 static void SDMMCHOST_SendCardActive (sdmmchost_t * *host*) [inline], [static]

Parameters

<i>host</i>	host handler
-------------	--------------

53.6.4.5.3 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.
---------------	-------------------------------

53.6.4.5.4 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.
-------------	-------------------------

53.6.4.5.5 `static uint32_t SDMMCHOST_GetSignalLineStatus (sdmmchost_t * host, uint32_t signalLine) [inline], [static]`

Parameters

<i>host</i>	host handler
<i>signalLine</i>	signal line type, reference <code>_sdmmc_signal_line</code>

53.6.4.5.6 `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.

53.6.4.5.7 `static void SDMMCHOST_EnableDDRMode (sdmmchost_t * host, bool enable, uint32_t nibblePos) [inline], [static]`

Parameters

<i>host</i>	host handler
<i>enable</i>	true is enable, false is disable.
<i>nibblePos</i>	nibble position indication. 0- the sequence is 'odd high nibble -> even high nibble -> odd low nibble -> even low nibble'; 1- the sequence is 'odd high nibble -> odd low nibble -> even high nibble -> even low nibble'.

53.6.4.5.8 `static void SDMMCHOST_EnableHS400Mode (sdmmchost_t * host, bool enable) [inline], [static]`

Parameters

<i>host</i>	host handler
<i>enable</i>	true is enable, false is disable.

53.6.4.5.9 `static void SDMMCHOST_EnableStrobeDII (sdmmchost_t * host, bool enable)`
`[inline], [static]`

Parameters

<i>host</i>	host handler
<i>enable</i>	true is enable, false is disable.

53.6.4.5.10 `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

53.6.4.5.11 `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

53.6.4.5.12 `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

53.6.4.5.13 `status_t SDMMCHOST_CardIntInit (sdmmchost_t * host, void * sdiInt)`

Parameters

<i>host</i>	host handler
<i>sdiInt</i>	card interrupt configuration

53.6.4.5.14 `static void SDMMCHOST_ForceClockOn (sdmmchost_t * host, bool enable)`
`[inline], [static]`

Parameters

<i>host</i>	host handler
<i>enable</i>	true is enable, false is disable.

53.6.4.5.15 `void SDMMCHOST_SwitchToVoltage (sdmmchost_t * host, uint32_t voltage)`

Parameters

<i>host</i>	host handler
<i>voltage</i>	switch to voltage level.

53.6.4.5.16 `status_t SDMMCHOST_CardDetectInit (sdmmchost_t * host, void * cd)`

Parameters

<i>host</i>	host handler
-------------	--------------

<i>cd</i>	card detect configuration
-----------	---------------------------

53.6.4.5.17 **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

53.6.4.5.18 **uint32_t SDMMCHOST_CardDetectStatus (sdmmchost_t * *host*)**

Parameters

<i>host</i>	host handler
-------------	--------------

Return values

<i>kSD_Inserted, kSD_Removed</i>	
----------------------------------	--

53.6.4.5.19 **status_t SDMMCHOST_Init (sdmmchost_t * *host*)**

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
* SDMMCHOST_Deinit (host);
* SDMMCHOST_Init (host);
*
```

Parameters

<i>host</i>	host handler
-------------	--------------

Return values

<i>kStatus_Success</i>	host init success
<i>kStatus_Fail</i>	event fail

53.6.4.5.20 void SDMMCHOST_Deinit (sdmmchost_t * *host*)

Please note it is a thread safe function.

Parameters

<i>host</i>	host handler
-------------	--------------

53.6.4.5.21 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.

53.6.4.5.22 status_t SDMMCHOST_TransferFunction (sdmmchost_t * *host*, sdmmchost_transfer_t * *content*)

Please note it is a thread safe function.

Note

the host transfer function support below functionality,

1. Non-cache line size alignment check on the data buffer, it is means that no matter the data buffer used for data transfer is align with cache line size or not, sdmmc host driver will use the address directly.
2. Cache line size alignment check on the data buffer, sdmmc host driver will check the data buffer address, if the buffer is not align with cache line size, sdmmc host driver will convert it to cache line size align buffer, the functionality is enabled by #define SDMMCHOST_ENABLE_CACHE_LINE_ALIGN_TRANSFER 1 #define FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER 1U If application would like to enable the cache line size align functionality, please make sure the SDMMCHOST_InstallCacheAlignBuffer is called before

submit data transfer request and make sure the installing buffer size is not smaller than 2 * cache line size.

Parameters

<i>host</i>	host handler
<i>content</i>	transfer content.

53.6.4.5.23 `status_t SDMMCHOST_ExecuteTuning (sdmmchost_t * host, uint32_t tuningCmd, uint32_t * revBuf, uint32_t blockSize)`

Parameters

<i>host</i>	host handler
<i>tuningCmd</i>	tuning command.
<i>revBuf</i>	receive buffer pointer
<i>blockSize</i>	tuning data block size.

53.6.4.5.24 `void SDMMCHOST_Reset (sdmmchost_t * host)`

Parameters

<i>host</i>	host handler
-------------	--------------

53.6.4.5.25 `void SDMMCHOST_ConvertDataToLittleEndian (sdmmchost_t * host, uint32_t * data, uint32_t wordSize, uint32_t format)`

Parameters

<i>host</i>	host handler.
<i>data</i>	data buffer address.
<i>wordSize</i>	data buffer size in word.
<i>format</i>	data packet format.

53.7 SDMMC Common

53.7.1 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** FSL_FEATURE_L1DCACHE_LINESIZE_BYTE
 - make sure the internal buffer address is cache align*
- #define **FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE** (FSL_SDMMC_DEFAULT_BLOCK_SIZE + SDMMC_DATA_BUFFER_ALIGN_CACHE)
 - sdmmc card internal buffer size*
- #define **FSL_SDMMC_CARD_MAX_BUS_FREQ**(max, target) ((max) == 0U ? (target) : ((max) > (target) ? (target) : (max)))
 - get maximum freq*
- #define **SDMMC_LOG**(format,...)
 - SD/MMC error log.*
- #define **SDMMC_CLOCK_400KHZ** (400000U)
 - SD/MMC card initialization clock frequency.*
- #define **SD_CLOCK_25MHZ** (25000000U)
 - SD card bus frequency 1 in high-speed mode.*
- #define **SD_CLOCK_50MHZ** (50000000U)
 - SD card bus frequency 2 in high-speed mode.*
- #define **SD_CLOCK_100MHZ** (100000000U)
 - SD card bus frequency in SDR50 mode.*
- #define **SD_CLOCK_208MHZ** (208000000U)
 - SD card bus frequency in SDR104 mode.*
- #define **MMC_CLOCK_26MHZ** (26000000U)
 - MMC card bus frequency 1 in high-speed mode.*
- #define **MMC_CLOCK_52MHZ** (52000000U)
 - MMC card bus frequency 2 in high-speed mode.*
- #define **MMC_CLOCK_DDR52** (52000000U)
 - MMC card bus frequency in high-speed DDR52 mode.*
- #define **MMC_CLOCK_HS200** (200000000U)
 - MMC card bus frequency in high-speed HS200 mode.*
- #define **MMC_CLOCK_HS400** (400000000U)
 - MMC card bus frequency in high-speed HS400 mode.*
- #define **SDMMC_MASK**(bit) (1UL << (bit))
 - mask convert*
- #define **SDMMC_R1_ALL_ERROR_FLAG**
 - R1 all the error flag.*
- #define **SDMMC_R1_CURRENT_STATE**(x) (((x)&0x00001E00U) >> 9U)
 - R1: current state.*
- #define **SDSPI_R7_VERSION_SHIFT** (28U)
 - The bit mask for COMMAND VERSION field in R7.*
- #define **SDSPI_R7_VERSION_MASK** (0xFU)
 - The bit mask for COMMAND VERSION field in R7.*
- #define **SDSPI_R7_VOLTAGE_SHIFT** (8U)
 - The bit shift for VOLTAGE ACCEPTED field in R7.*
- #define **SDSPI_R7_VOLTAGE_MASK** (0xFU)

- *The bit mask for VOLTAGE ACCEPTED field in R7.*
#define **SDSPI_R7_VOLTAGE_27_36_MASK** (0x1U << SDSPI_R7_VOLTAGE_SHIFT)
- *The bit mask for VOLTAGE 2.7V to 3.6V field in R7.*
#define **SDSPI_R7_ECHO_SHIFT** (0U)
- *The bit shift for ECHO field in R7.*
#define **SDSPI_R7_ECHO_MASK** (0xFFU)
- *The bit mask for ECHO field in R7.*
#define **SDSPI_DATA_ERROR_TOKEN_MASK** (0xFU)
- *Data error token mask.*
#define **SDSPI_DATA_RESPONSE_TOKEN_MASK** (0x1FU)
- *Mask for data response bits.*
#define **SDIO_CCCR_REG_NUMBER** (0x16U)
- *sdio card cccr register number*
#define **SDIO_IO_READY_TIMEOUT_UNIT** (10U)
- *sdio IO ready timeout steps*
#define **SDIO_CMD_ARGUMENT_RW_POS** (31U)
- *read/write flag position*
#define **SDIO_CMD_ARGUMENT_FUNC_NUM_POS** (28U)
- *function number position*
#define **SDIO_DIRECT_CMD_ARGUMENT_RAW_POS** (27U)
- *direct raw flag position*
#define **SDIO_CMD_ARGUMENT_REG_ADDR_POS** (9U)
- *direct reg addr position*
#define **SDIO_CMD_ARGUMENT_REG_ADDR_MASK** (0x1FFFFU)
- *direct reg addr mask*
#define **SDIO_DIRECT_CMD_DATA_MASK** (0xFFU)
- *data mask*
#define **SDIO_EXTEND_CMD_ARGUMENT_BLOCK_MODE_POS** (27U)
- *extended command argument block mode bit position*
#define **SDIO_EXTEND_CMD_ARGUMENT_OP_CODE_POS** (26U)
- *extended command argument OP Code bit position*
#define **SDIO_EXTEND_CMD_BLOCK_MODE_MASK** (0x08000000U)
- *block mode mask*
#define **SDIO_EXTEND_CMD_OP_CODE_MASK** (0x04000000U)
- *op code mask*
#define **SDIO_EXTEND_CMD_COUNT_MASK** (0x1FFU)
- *byte/block count mask*
#define **SDIO_MAX_BLOCK_SIZE** (2048U)
- *max block size*
#define **SDIO_FBR_BASE(x)** ((x)*0x100U)
- *function basic register*
#define **SDIO_TPL_CODE_END** (0xFFU)
- *tuple end*
#define **SDIO_TPL_CODE_MANIFID** (0x20U)
- *manufacturer ID*
#define **SDIO_TPL_CODE_FUNCID** (0x21U)
- *function ID*
#define **SDIO_TPL_CODE_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` (1UL << 9U)
UHS timing mode flag.
- #define `SDIO_CCCR_DRIVER_TYPE_MASK` (3U << 4U)
Driver type flag.
- #define `SDIO_CCCR_ASYNC_INT_MASK` (1U)
async interrupt flag
- #define `SDIO_CCCR_SUPPORT_8BIT_BUS` (1UL << 18U)
8 bit data bus flag
- #define `MMC_OCR_V170TO195_SHIFT` (7U)
The bit mask for VOLTAGE WINDOW 1.70V to 1.95V field in OCR.
- #define `MMC_OCR_V170TO195_MASK` (0x00000080U)
The bit mask for VOLTAGE WINDOW 1.70V to 1.95V field in OCR.
- #define `MMC_OCR_V200TO260_SHIFT` (8U)
The bit shift for VOLTAGE WINDOW 2.00V to 2.60V field in OCR.
- #define `MMC_OCR_V200TO260_MASK` (0x00007F00U)
The bit mask for VOLTAGE WINDOW 2.00V to 2.60V field in OCR.
- #define `MMC_OCR_V270TO360_SHIFT` (15U)
The bit shift for VOLTAGE WINDOW 2.70V to 3.60V field in OCR.
- #define `MMC_OCR_V270TO360_MASK` (0x00FF8000U)
The bit mask for VOLTAGE WINDOW 2.70V to 3.60V field in OCR.
- #define `MMC_OCR_ACCESS_MODE_SHIFT` (29U)
The bit shift for ACCESS MODE field in OCR.
- #define `MMC_OCR_ACCESS_MODE_MASK` (0x60000000U)
The bit mask for ACCESS MODE field in OCR.
- #define `MMC_OCR_BUSY_SHIFT` (31U)
The bit shift for BUSY field in OCR.
- #define `MMC_OCR_BUSY_MASK` (1U << MMC_OCR_BUSY_SHIFT)
The bit mask for BUSY field in OCR.
- #define `MMC_TRANSFER_SPEED_FREQUENCY_UNIT_SHIFT` (0U)
The bit shift for FREQUENCY UNIT field in TRANSFER SPEED(TRAN-SPEED in Extended CSD)
- #define `MMC_TRANSFER_SPEED_FREQUENCY_UNIT_MASK` (0x07U)
The bit mask for 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_CACHE_CONTROL_ENABLE` (1U)

- *mmc cache control enable*
- #define `MMC_CACHE_TRIGGER_FLUSH` (1U)
- *mmc cache flush*
- #define `MMC_DATA_BUS_WIDTH_TYPE_NUMBER` (3U)
- *The number of data bus width type.*
- #define `MMC_PARTITION_CONFIG_PARTITION_ACCESS_SHIFT` (0U)
- *The bit shift for PARTITION ACCESS 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` (2UL)
- *MMC card default relative address.*
- #define `SD_PRODUCT_NAME_BYTES` (5U)
- *SD card product name length united as bytes.*
- #define `SD_AU_START_VALUE` (1U)
- *SD AU start value.*
- #define `SD_UHS_AU_START_VALUE` (7U)
- *SD UHS AU start value.*
- #define `SD_TRANSFER_SPEED_RATE_UNIT_SHIFT` (0U)
- *The bit shift for RATE UNIT field in TRANSFER SPEED.*
- #define `SD_TRANSFER_SPEED_RATE_UNIT_MASK` (0x07U)
- *The bit mask for RATE UNIT field in TRANSFER SPEED.*
- #define `SD_TRANSFER_SPEED_TIME_VALUE_SHIFT` (2U)
- *The bit shift for TIME VALUE field in TRANSFER SPEED.*
- #define `SD_TRANSFER_SPEED_TIME_VALUE_MASK` (0x78U)
- *The bit mask for TIME VALUE field in TRANSFER SPEED.*
- #define `SD_RD_TRANSFER_SPEED_RATE_UNIT(x)` (((x.transferSpeed) & `SD_TRANSFER_SPEED_RATE_UNIT_MASK`) >> `SD_TRANSFER_SPEED_RATE_UNIT_SHIFT`)
- *Read the value of FREQUENCY UNIT in TRANSFER SPEED field.*
- #define `SD_RD_TRANSFER_SPEED_TIME_VALUE(x)` (((x.transferSpeed) & `SD_TRANSFER-`

`_SPEED_TIME_VALUE_MASK) >> SD_TRANSFER_SPEED_TIME_VALUE_SHIFT)`

Read the value of TIME VALUE in TRANSFER SPEED field.

- #define `MMC_PRODUCT_NAME_BYTES` (6U)
MMC card product name length united as bytes.
- #define `MMC_SWITCH_COMMAND_SET_SHIFT` (0U)
The bit shift for COMMAND SET field in SWITCH command.
- #define `MMC_SWITCH_COMMAND_SET_MASK` (0x00000007U)
The bit mask for COMMAND set field in SWITCH command.
- #define `MMC_SWITCH_VALUE_SHIFT` (8U)
The bit shift for VALUE field in SWITCH command.
- #define `MMC_SWITCH_VALUE_MASK` (0x0000FF00U)
The bit mask for VALUE field in SWITCH command.
- #define `MMC_SWITCH_BYTE_INDEX_SHIFT` (16U)
The bit shift for BYTE INDEX field in SWITCH command.
- #define `MMC_SWITCH_BYTE_INDEX_MASK` (0x00FF0000U)
The bit mask for BYTE INDEX field in SWITCH command.
- #define `MMC_SWITCH_ACCESS_MODE_SHIFT` (24U)
The bit shift for ACCESS MODE field in SWITCH command.
- #define `MMC_SWITCH_ACCESS_MODE_MASK` (0x03000000U)
The bit mask for ACCESS MODE field in SWITCH command.

Typedefs

- typedef void(* `sd_cd_t`)(bool isInserted, void *userData)
card detect application callback definition
- typedef bool(* `sd_cd_status_t`)(void)
card detect status
- typedef void(* `sd_io_voltage_func_t`)(sdmmc_operation_voltage_t voltage)
card switch voltage function pointer
- typedef void(* `sd_pwr_t`)(bool enable)
card power control function pointer
- typedef void(* `sd_io_strength_t`)(uint32_t busFreq)
card io strength control
- typedef void(* `sdio_int_t`)(void *userData)
card interrupt function pointer

Enumerations

- enum {
 - kStatus_SDMMC_NotSupportYet = MAKE_STATUS(kStatusGroup_SDMMC, 0U),
 - kStatus_SDMMC_TransferFailed = MAKE_STATUS(kStatusGroup_SDMMC, 1U),
 - kStatus_SDMMC_SetCardBlockSizeFailed = MAKE_STATUS(kStatusGroup_SDMMC, 2U),
 - kStatus_SDMMC_HostNotSupport = MAKE_STATUS(kStatusGroup_SDMMC, 3U),
 - kStatus_SDMMC_CardNotSupport = MAKE_STATUS(kStatusGroup_SDMMC, 4U),
 - kStatus_SDMMC_AllSendCidFailed = MAKE_STATUS(kStatusGroup_SDMMC, 5U),
 - kStatus_SDMMC_SendRelativeAddressFailed = MAKE_STATUS(kStatusGroup_SDMMC, 6U),
 - kStatus_SDMMC_SendCsdFailed = MAKE_STATUS(kStatusGroup_SDMMC, 7U),
 - kStatus_SDMMC_SelectCardFailed = MAKE_STATUS(kStatusGroup_SDMMC, 8U),
 - kStatus_SDMMC_SendScrFailed = MAKE_STATUS(kStatusGroup_SDMMC, 9U),
 - kStatus_SDMMC_SetDataBusWidthFailed = MAKE_STATUS(kStatusGroup_SDMMC, 10U),
 - kStatus_SDMMC_GoIdleFailed = MAKE_STATUS(kStatusGroup_SDMMC, 11U),
 - kStatus_SDMMC_HandShakeOperationConditionFailed,
 - kStatus_SDMMC_SendApplicationCommandFailed,
 - kStatus_SDMMC_SwitchFailed = MAKE_STATUS(kStatusGroup_SDMMC, 14U),
 - kStatus_SDMMC_StopTransmissionFailed = MAKE_STATUS(kStatusGroup_SDMMC, 15U),
 - kStatus_SDMMC_WaitWriteCompleteFailed = MAKE_STATUS(kStatusGroup_SDMMC, 16U),
 - kStatus_SDMMC_SetBlockCountFailed = MAKE_STATUS(kStatusGroup_SDMMC, 17U),
 - kStatus_SDMMC_SetRelativeAddressFailed = MAKE_STATUS(kStatusGroup_SDMMC, 18U),
 - kStatus_SDMMC_SwitchBusTimingFailed = MAKE_STATUS(kStatusGroup_SDMMC, 19U),
 - kStatus_SDMMC_SendExtendedCsdFailed = MAKE_STATUS(kStatusGroup_SDMMC, 20U),
 - kStatus_SDMMC_ConfigureBootFailed = MAKE_STATUS(kStatusGroup_SDMMC, 21U),
 - kStatus_SDMMC_ConfigureExtendedCsdFailed = MAKE_STATUS(kStatusGroup_SDMMC, 22-
U),
 - kStatus_SDMMC_EnableHighCapacityEraseFailed,
 - kStatus_SDMMC_SendTestPatternFailed = MAKE_STATUS(kStatusGroup_SDMMC, 24U),
 - kStatus_SDMMC_ReceiveTestPatternFailed = MAKE_STATUS(kStatusGroup_SDMMC, 25U),
 - kStatus_SDMMC_SDIO_ResponseError = MAKE_STATUS(kStatusGroup_SDMMC, 26U),
 - kStatus_SDMMC_SDIO_InvalidArgument,
 - kStatus_SDMMC_SDIO_SendOperationConditionFail,
 - kStatus_SDMMC_InvalidVoltage = MAKE_STATUS(kStatusGroup_SDMMC, 29U),
 - kStatus_SDMMC_SDIO_SwitchHighSpeedFail = MAKE_STATUS(kStatusGroup_SDMMC, 30-
U),
 - kStatus_SDMMC_SDIO_ReadCISFail = MAKE_STATUS(kStatusGroup_SDMMC, 31U),
 - kStatus_SDMMC_SDIO_InvalidCard = MAKE_STATUS(kStatusGroup_SDMMC, 32U),
 - kStatus_SDMMC_TuningFail = MAKE_STATUS(kStatusGroup_SDMMC, 33U),
 - kStatus_SDMMC_SwitchVoltageFail = MAKE_STATUS(kStatusGroup_SDMMC, 34U),
 - kStatus_SDMMC_SwitchVoltage18VFail33VSuccess = MAKE_STATUS(kStatusGroup_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),
kStatus_SDMMC_PollingCardIdleFailed = MAKE_STATUS(kStatusGroup_SDMMC, 42U),
kStatus_SDMMC_DeselectCardFailed = MAKE_STATUS(kStatusGroup_SDMMC, 43U),
kStatus_SDMMC_CardStatusIdle = MAKE_STATUS(kStatusGroup_SDMMC, 44U),
kStatus_SDMMC_CardStatusBusy = MAKE_STATUS(kStatusGroup_SDMMC, 45U),
kStatus_SDMMC_CardInitFailed = MAKE_STATUS(kStatusGroup_SDMMC, 46U) }

```

SD/MMC card API's running status.

- enum {


```

kSDMMC_SignalLineCmd = 1U,
kSDMMC_SignalLineData0 = 2U,
kSDMMC_SignalLineData1 = 4U,
kSDMMC_SignalLineData2 = 8U,
kSDMMC_SignalLineData3 = 16U,
kSDMMC_SignalLineData4 = 32U,
kSDMMC_SignalLineData5 = 64U,
kSDMMC_SignalLineData6 = 128U,
kSDMMC_SignalLineData7 = 256U }

```

sdmmc signal line
- enum sdmmc_operation_voltage_t {


```

kSDMMC_OperationVoltageNone = 0U,
kSDMMC_OperationVoltage330V = 1U,
kSDMMC_OperationVoltage300V = 2U,
kSDMMC_OperationVoltage180V = 3U }

```

card operation voltage
- enum {


```

kSDMMC_BusWidth1Bit = 0U,
kSDMMC_BusWidth4Bit = 1U,
kSDMMC_BusWidth8Bit = 2U }

```

card bus width
- enum { kSDMMC_Support8BitWidth = 1U }

sdmmc capability flag
- enum {


```

kSDMMC_DataPacketFormatLSBFirst,
kSDMMC_DataPacketFormatMSBFirst }

```

@ brief sdmmc data packet format
- enum sd_detect_card_type_t {


```

kSD_DetectCardByGpioCD,
kSD_DetectCardByHostCD,
kSD_DetectCardByHostDATA3 }

```

sd card detect type

- enum {
 - kSD_Inserted = 1U,
 - kSD_Removed = 0U }
 - @ brief SD card detect status*
- enum {
 - kSD_DAT3PullDown = 0U,
 - kSD_DAT3PullUp = 1U }
 - @ brief SD card detect status*
- enum sd_io_voltage_ctrl_type_t {
 - kSD_IOVoltageCtrlNotSupport = 0U,
 - kSD_IOVoltageCtrlByGpio = 2U }
 - io voltage control type*
- enum {
 - kSDMMC_R1OutOfRangeFlag = 31,
 - kSDMMC_R1AddressErrorFlag = 30,
 - kSDMMC_R1BlockLengthErrorFlag = 29,
 - kSDMMC_R1EraseSequenceErrorFlag = 28,
 - kSDMMC_R1EraseParameterErrorFlag = 27,
 - kSDMMC_R1WriteProtectViolationFlag = 26,
 - kSDMMC_R1CardIsLockedFlag = 25,
 - kSDMMC_R1LockUnlockFailedFlag = 24,
 - kSDMMC_R1CommandCrcErrorFlag = 23,
 - kSDMMC_R1IllegalCommandFlag = 22,
 - kSDMMC_R1CardEccFailedFlag = 21,
 - kSDMMC_R1CardControllerErrorFlag = 20,
 - kSDMMC_R1ErrorFlag = 19,
 - kSDMMC_R1CidCsdOverwriteFlag = 16,
 - kSDMMC_R1WriteProtectEraseSkipFlag = 15,
 - kSDMMC_R1CardEccDisabledFlag = 14,
 - kSDMMC_R1EraseResetFlag = 13,
 - kSDMMC_R1ReadyForDataFlag = 8,
 - kSDMMC_R1SwitchErrorFlag = 7,
 - kSDMMC_R1ApplicationCommandFlag = 5,
 - kSDMMC_R1AuthenticationSequenceErrorFlag = 3 }
 - Card status bit in R1.*
- enum sdmmc_r1_current_state_t {
 - kSDMMC_R1StateIdle = 0U,
 - kSDMMC_R1StateReady = 1U,
 - kSDMMC_R1StateIdentify = 2U,
 - kSDMMC_R1StateStandby = 3U,
 - kSDMMC_R1StateTransfer = 4U,
 - kSDMMC_R1StateSendData = 5U,
 - kSDMMC_R1StateReceiveData = 6U,
 - kSDMMC_R1StateProgram = 7U,
 - kSDMMC_R1StateDisconnect = 8U }
 - CURRENT_STATE filed in R1.*

- enum {
 - kSDSPI_R1InIdleStateFlag = (1U << 0U),
 - kSDSPI_R1EraseResetFlag = (1U << 1U),
 - kSDSPI_R1IllegalCommandFlag = (1U << 2U),
 - kSDSPI_R1CommandCrcErrorFlag = (1U << 3U),
 - kSDSPI_R1EraseSequenceErrorFlag = (1U << 4U),
 - kSDSPI_R1AddressErrorFlag = (1U << 5U),
 - kSDSPI_R1ParameterErrorFlag = (1U << 6U) }

Error bit in SPI mode R1.
- enum {
 - kSDSPI_R2CardLockedFlag = (1U << 0U),
 - kSDSPI_R2WriteProtectEraseSkip = (1U << 1U),
 - kSDSPI_R2LockUnlockFailed = (1U << 1U),
 - kSDSPI_R2ErrorFlag = (1U << 2U),
 - kSDSPI_R2CardControllerErrorFlag = (1U << 3U),
 - kSDSPI_R2CardEccFailedFlag = (1U << 4U),
 - kSDSPI_R2WriteProtectViolationFlag = (1U << 5U),
 - kSDSPI_R2EraseParameterErrorFlag = (1U << 6U),
 - kSDSPI_R2OutOfRangeFlag = (1U << 7U),
 - kSDSPI_R2CsdOverwriteFlag = (1U << 7U) }

Error bit in SPI mode R2.
- enum {
 - kSDSPI_DataErrorTokenError = (1U << 0U),
 - kSDSPI_DataErrorTokenCardControllerError = (1U << 1U),
 - kSDSPI_DataErrorTokenCardEccFailed = (1U << 2U),
 - kSDSPI_DataErrorTokenOutOfRange = (1U << 3U) }

Data Error Token mask bit.
- enum sdspi_data_token_t {
 - kSDSPI_DataTokenBlockRead = 0xFEU,
 - kSDSPI_DataTokenSingleBlockWrite = 0xFEU,
 - kSDSPI_DataTokenMultipleBlockWrite = 0xFCU,
 - kSDSPI_DataTokenStopTransfer = 0xFDU }

Data Token.
- enum sdspi_data_response_token_t {
 - kSDSPI_DataResponseTokenAccepted = 0x05U,
 - kSDSPI_DataResponseTokenCrcError = 0x0BU,
 - kSDSPI_DataResponseTokenWriteError = 0x0DU }

Data Response Token.
- enum sd_command_t {
 - kSD_SendRelativeAddress = 3U,
 - kSD_Switch = 6U,
 - kSD_SendInterfaceCondition = 8U,
 - kSD_VoltageSwitch = 11U,
 - kSD_SpeedClassControl = 20U,
 - kSD_EraseWriteBlockStart = 32U,
 - kSD_EraseWriteBlockEnd = 33U,

`kSD_SendTuningBlock = 19U }`

SD card individual commands.

- `enum sdsapi_command_t { kSDSPI_CommandCrc = 59U }`

SDSPI individual commands.

- `enum sd_application_command_t {`
`kSD_ApplicationSetBusWidth = 6U,`
`kSD_ApplicationStatus = 13U,`
`kSD_ApplicationSendNumberWriteBlocks = 22U,`
`kSD_ApplicationSetWriteBlockEraseCount = 23U,`
`kSD_ApplicationSendOperationCondition = 41U,`
`kSD_ApplicationSetClearCardDetect = 42U,`
`kSD_ApplicationSendScr = 51U }`

SD card individual application commands.

- `enum {`
`kSDMMC_CommandClassBasic = (1U << 0U),`
`kSDMMC_CommandClassBlockRead = (1U << 2U),`
`kSDMMC_CommandClassBlockWrite = (1U << 4U),`
`kSDMMC_CommandClassErase = (1U << 5U),`
`kSDMMC_CommandClassWriteProtect = (1U << 6U),`
`kSDMMC_CommandClassLockCard = (1U << 7U),`
`kSDMMC_CommandClassApplicationSpecific = (1U << 8U),`
`kSDMMC_CommandClassInputOutputMode = (1U << 9U),`
`kSDMMC_CommandClassSwitch = (1U << 10U) }`

SD card command class.

- `enum {`
`kSD_OcrPowerUpBusyFlag = 31,`
`kSD_OcrHostCapacitySupportFlag = 30,`
`kSD_OcrCardCapacitySupportFlag = kSD_OcrHostCapacitySupportFlag,`
`kSD_OcrSwitch18RequestFlag = 24,`
`kSD_OcrSwitch18AcceptFlag = kSD_OcrSwitch18RequestFlag,`
`kSD_OcrVdd27_28Flag = 15,`
`kSD_OcrVdd28_29Flag = 16,`
`kSD_OcrVdd29_30Flag = 17,`
`kSD_OcrVdd30_31Flag = 18,`
`kSD_OcrVdd31_32Flag = 19,`
`kSD_OcrVdd32_33Flag = 20,`
`kSD_OcrVdd33_34Flag = 21,`
`kSD_OcrVdd34_35Flag = 22,`
`kSD_OcrVdd35_36Flag = 23 }`

OCR register in SD card.

- `enum {`
`kSD_SpecificationVersion1_0 = (1U << 0U),`
`kSD_SpecificationVersion1_1 = (1U << 1U),`
`kSD_SpecificationVersion2_0 = (1U << 2U),`
`kSD_SpecificationVersion3_0 = (1U << 3U) }`

SD card specification version number.

- enum `sd_switch_mode_t` {
`kSD_SwitchCheck = 0U`,
`kSD_SwitchSet = 1U` }
SD card switch mode.
- enum {
`kSD_CsdReadBlockPartialFlag = (1U << 0U)`,
`kSD_CsdWriteBlockMisalignFlag = (1U << 1U)`,
`kSD_CsdReadBlockMisalignFlag = (1U << 2U)`,
`kSD_CsdDsriImplementedFlag = (1U << 3U)`,
`kSD_CsdEraseBlockEnabledFlag = (1U << 4U)`,
`kSD_CsdWriteProtectGroupEnabledFlag = (1U << 5U)`,
`kSD_CsdWriteBlockPartialFlag = (1U << 6U)`,
`kSD_CsdFileFormatGroupFlag = (1U << 7U)`,
`kSD_CsdCopyFlag = (1U << 8U)`,
`kSD_CsdPermanentWriteProtectFlag = (1U << 9U)`,
`kSD_CsdTemporaryWriteProtectFlag = (1U << 10U)` }
SD card CSD register flags.
- enum {
`kSD_ScrDataStatusAfterErase = (1U << 0U)`,
`kSD_ScrSdSpecification3 = (1U << 1U)` }
SD card SCR register flags.
- enum {
`kSD_FunctionSDR12Default = 0U`,
`kSD_FunctionSDR25HighSpeed = 1U`,
`kSD_FunctionSDR50 = 2U`,
`kSD_FunctionSDR104 = 3U`,
`kSD_FunctionDDR50 = 4U` }
SD timing function number.
- enum {
`kSD_GroupTimingMode = 0U`,
`kSD_GroupCommandSystem = 1U`,
`kSD_GroupDriverStrength = 2U`,
`kSD_GroupCurrentLimit = 3U` }
SD group number.
- enum `sd_timing_mode_t` {
`kSD_TimingSDR12DefaultMode = 0U`,
`kSD_TimingSDR25HighSpeedMode = 1U`,
`kSD_TimingSDR50Mode = 2U`,
`kSD_TimingSDR104Mode = 3U`,
`kSD_TimingDDR50Mode = 4U` }
SD card timing mode flags.
- enum `sd_driver_strength_t` {
`kSD_DriverStrengthTypeB = 0U`,
`kSD_DriverStrengthTypeA = 1U`,
`kSD_DriverStrengthTypeC = 2U`,
`kSD_DriverStrengthTypeD = 3U` }

- SD card driver strength.*
 - enum `sd_max_current_t` {
 - `kSD_CurrentLimit200MA` = 0U,
 - `kSD_CurrentLimit400MA` = 1U,
 - `kSD_CurrentLimit600MA` = 2U,
 - `kSD_CurrentLimit800MA` = 3U }
 - SD card current limit.*
 - enum `sdmmc_command_t` {
 - `kSDMMC_GoIdleState` = 0U,
 - `kSDMMC_AllSendCid` = 2U,
 - `kSDMMC_SetDsr` = 4U,
 - `kSDMMC_SelectCard` = 7U,
 - `kSDMMC_SendCsd` = 9U,
 - `kSDMMC_SendCid` = 10U,
 - `kSDMMC_StopTransmission` = 12U,
 - `kSDMMC_SendStatus` = 13U,
 - `kSDMMC_GoInactiveState` = 15U,
 - `kSDMMC_SetBlockLength` = 16U,
 - `kSDMMC_ReadSingleBlock` = 17U,
 - `kSDMMC_ReadMultipleBlock` = 18U,
 - `kSDMMC_SetBlockCount` = 23U,
 - `kSDMMC_WriteSingleBlock` = 24U,
 - `kSDMMC_WriteMultipleBlock` = 25U,
 - `kSDMMC_ProgramCsd` = 27U,
 - `kSDMMC_SetWriteProtect` = 28U,
 - `kSDMMC_ClearWriteProtect` = 29U,
 - `kSDMMC_SendWriteProtect` = 30U,
 - `kSDMMC_Erase` = 38U,
 - `kSDMMC_LockUnlock` = 42U,
 - `kSDMMC_ApplicationCommand` = 55U,
 - `kSDMMC_GeneralCommand` = 56U,
 - `kSDMMC_ReadOcr` = 58U }
 - SD/MMC card common commands.*
 - enum {

- ```

kSDIO_RegCCCRSdioVer = 0x00U,
kSDIO_RegSDVersion = 0x01U,
kSDIO_RegIOEnable = 0x02U,
kSDIO_RegIOReady = 0x03U,
kSDIO_RegIOIntEnable = 0x04U,
kSDIO_RegIOIntPending = 0x05U,
kSDIO_RegIOAbort = 0x06U,
kSDIO_RegBusInterface = 0x07U,
kSDIO_RegCardCapability = 0x08U,
kSDIO_RegCommonCISPointer = 0x09U,
kSDIO_RegBusSuspend = 0x0C,
kSDIO_RegFunctionSelect = 0x0DU,
kSDIO_RegExecutionFlag = 0x0EU,
kSDIO_RegReadyFlag = 0x0FU,
kSDIO_RegFN0BlockSizeLow = 0x10U,
kSDIO_RegFN0BlockSizeHigh = 0x11U,
kSDIO_RegPowerControl = 0x12U,
kSDIO_RegBusSpeed = 0x13U,
kSDIO_RegUHSITimingSupport = 0x14U,
kSDIO_RegDriverStrength = 0x15U,
kSDIO_RegInterruptExtension = 0x16U }

```
- sdio card cccr register addr*

    - enum `sdio_command_t` {

```

kSDIO_SendRelativeAddress = 3U,
kSDIO_SendOperationCondition = 5U,
kSDIO_SendInterfaceCondition = 8U,
kSDIO_RWIODirect = 52U,
kSDIO_RWIOExtended = 53U }

```

*sdio card individual commands*

      - enum `sdio_func_num_t` {

```

kSDIO_FunctionNum0,
kSDIO_FunctionNum1,
kSDIO_FunctionNum2,
kSDIO_FunctionNum3,
kSDIO_FunctionNum4,
kSDIO_FunctionNum5,
kSDIO_FunctionNum6,
kSDIO_FunctionNum7,
kSDIO_FunctionMemory }

```

*sdio card individual commands*

        - enum {



```

kSDIO_StatusCmdCRCError = 0x8000U,
kSDIO_StatusIllegalCmd = 0x4000U,
kSDIO_StatusR6Error = 0x2000U,
kSDIO_StatusError = 0x0800U,
kSDIO_StatusFunctionNumError = 0x0200U,
kSDIO_StatusOutOfRange = 0x0100U }
 sdio command response flag
• enum {
kSDIO_OcrPowerUpBusyFlag = 31,
kSDIO_OcrIONumber = 28,
kSDIO_OcrMemPresent = 27,
kSDIO_OcrVdd20_21Flag = 8,
kSDIO_OcrVdd21_22Flag = 9,
kSDIO_OcrVdd22_23Flag = 10,
kSDIO_OcrVdd23_24Flag = 11,
kSDIO_OcrVdd24_25Flag = 12,
kSDIO_OcrVdd25_26Flag = 13,
kSDIO_OcrVdd26_27Flag = 14,
kSDIO_OcrVdd27_28Flag = 15,
kSDIO_OcrVdd28_29Flag = 16,
kSDIO_OcrVdd29_30Flag = 17,
kSDIO_OcrVdd30_31Flag = 18,
kSDIO_OcrVdd31_32Flag = 19,
kSDIO_OcrVdd32_33Flag = 20,
kSDIO_OcrVdd33_34Flag = 21,
kSDIO_OcrVdd34_35Flag = 22,
kSDIO_OcrVdd35_36Flag = 23 }
 sdio operation condition flag
• enum {
kSDIO_CCCRSupportDirectCmdDuringDataTrans = (1UL << 0U),
kSDIO_CCCRSupportMultiBlock = (1UL << 1U),
kSDIO_CCCRSupportReadWait = (1UL << 2U),
kSDIO_CCCRSupportSuspendResume = (1UL << 3U),
kSDIO_CCCRSupportIntDuring4BitDataTrans = (1UL << 4U),
kSDIO_CCCRSupportLowSpeed1Bit = (1UL << 6U),
kSDIO_CCCRSupportLowSpeed4Bit = (1UL << 7U),
kSDIO_CCCRSupportMasterPowerControl = (1UL << 8U),
kSDIO_CCCRSupportHighSpeed = (1UL << 9U),
kSDIO_CCCRSupportContinuousSPIInt = (1UL << 10U) }
 sdio capability flag
• enum {
kSDIO_FBRSupportCSA = (1U << 0U),
kSDIO_FBRSupportPowerSelection = (1U << 1U) }
 sdio fbr flag
• enum sdio_bus_width_t {

```



```
kSDIO_DataBus1Bit = 0x00U,
kSDIO_DataBus4Bit = 0x02U,
kSDIO_DataBus8Bit = 0x03U }
```

*sdio bus width*

- enum `mmc_command_t` {
 

```
kMMC_SendOperationCondition = 1U,
kMMC_SetRelativeAddress = 3U,
kMMC_SleepAwake = 5U,
kMMC_Switch = 6U,
kMMC_SendExtendedCsd = 8U,
kMMC_ReadDataUntilStop = 11U,
kMMC_BusTestRead = 14U,
kMMC_SendingBusTest = 19U,
kMMC_WriteDataUntilStop = 20U,
kMMC_SendTuningBlock = 21U,
kMMC_ProgramCid = 26U,
kMMC_EraseGroupStart = 35U,
kMMC_EraseGroupEnd = 36U,
kMMC_FastInputOutput = 39U,
kMMC_GoInterruptState = 40U }
```

*MMC card individual commands.*
- enum `mmc_classified_voltage_t` {
 

```
kMMC_ClassifiedVoltageHigh = 0U,
kMMC_ClassifiedVoltageDual = 1U }
```

*MMC card classified as voltage range.*
- enum `mmc_classified_density_t` { `kMMC_ClassifiedDensityWithin2GB = 0U` }
 

*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 {
  - kMMC\_ExtendedCsdRevision10 = 0U,
  - kMMC\_ExtendedCsdRevision11 = 1U,
  - kMMC\_ExtendedCsdRevision12 = 2U,
  - kMMC\_ExtendedCsdRevision13 = 3U,
  - kMMC\_ExtendedCsdRevision14 = 4U,
  - kMMC\_ExtendedCsdRevision15 = 5U,
  - kMMC\_ExtendedCsdRevision16 = 6U,
  - kMMC\_ExtendedCsdRevision17 = 7U }
  - MMC card Extended CSD fix version(EXT\_CSD\_REV in Extended CSD)*
- enum mmc\_command\_set\_t {
  - kMMC\_CommandSetStandard = 0U,
  - kMMC\_CommandSet1 = 1U,
  - kMMC\_CommandSet2 = 2U,
  - kMMC\_CommandSet3 = 3U,
  - kMMC\_CommandSet4 = 4U }
  - MMC card command set(COMMAND\_SET in Extended CSD)*
- enum {
  - kMMC\_SupportAlternateBoot = 1U,
  - kMMC\_SupportDDRBoot = 2U,
  - kMMC\_SupportHighSpeedBoot = 4U }
  - boot support(BOOT\_INFO in Extended CSD)*
- enum mmc\_high\_speed\_timing\_t {
  - kMMC\_HighSpeedTimingNone = 0U,
  - kMMC\_HighSpeedTiming = 1U,
  - kMMC\_HighSpeed200Timing = 2U,
  - kMMC\_HighSpeed400Timing = 3U,
  - kMMC\_EnhanceHighSpeed400Timing = 4U }
  - MMC card high-speed timing(HS\_TIMING in Extended CSD)*
- enum mmc\_data\_bus\_width\_t {
  - kMMC\_DataBusWidth1bit = 0U,
  - kMMC\_DataBusWidth4bit = 1U,
  - kMMC\_DataBusWidth8bit = 2U,
  - kMMC\_DataBusWidth4bitDDR = 5U,
  - kMMC\_DataBusWidth8bitDDR = 6U,
  - kMMC\_DataBusWidth8bitDDRSTROBE = 0x86U }
  - MMC card data bus width(BUS\_WIDTH in Extended CSD)*
- enum mmc\_boot\_partition\_enable\_t {

```

kMMC_BootPartitionEnableNot = 0U,
kMMC_BootPartitionEnablePartition1 = 1U,
kMMC_BootPartitionEnablePartition2 = 2U,
kMMC_BootPartitionEnableUserAera = 7U }

```

*MMC card boot partition enabled(BOOT\_PARTITION\_ENABLE in Extended CSD)*

- enum mmc\_boot\_timing\_mode\_t {

```

kMMC_BootModeSDRWithDefaultTiming = 0U,
kMMC_BootModeSDRWithHighSpeedTiming = 1U,
kMMC_BootModeDDRTiming = 2U }

```

*boot mode configuration Note: HS200 & HS400 is not support during BOOT operation.*

- enum mmc\_boot\_partition\_wp\_t {

```

kMMC_BootPartitionWPDisable = 0x50U,
kMMC_BootPartitionPwrWPToBothPartition,
kMMC_BootPartitionPermWPToBothPartition = 0x04U,
kMMC_BootPartitionPwrWPToPartition1 = (1U << 7U) | 1U,
kMMC_BootPartitionPwrWPToPartition2 = (1U << 7U) | 3U,
kMMC_BootPartitionPermWPToPartition1,
kMMC_BootPartitionPermWPToPartition2,
kMMC_BootPartitionPermWPToPartition1PwrWPToPartition2,
kMMC_BootPartitionPermWPToPartition2PwrWPToPartition1 }

```

*MMC card boot partition write protect configurations All the bits in BOOT\_WP register, except the two R/W bits B\_PERM\_WP\_DIS and B\_PERM\_WP\_EN, shall only be written once per power cycle. The protection mdde intended for both boot areas will be set with a single write.*

- enum {

```

kMMC_BootPartitionNotProtected = 0U,
kMMC_BootPartitionPwrProtected = 1U,
kMMC_BootPartitionPermProtected = 2U }

```

*MMC card boot partition write protect status.*

- enum mmc\_access\_partition\_t {

```

kMMC_AccessPartitionUserAera = 0U,
kMMC_AccessPartitionBoot1 = 1U,
kMMC_AccessPartitionBoot2 = 2U,
kMMC_AccessRPMB = 3U,
kMMC_AccessGeneralPurposePartition1 = 4U,
kMMC_AccessGeneralPurposePartition2 = 5U,
kMMC_AccessGeneralPurposePartition3 = 6U,
kMMC_AccessGeneralPurposePartition4 = 7U }

```

*MMC card partition to be accessed(BOOT\_PARTITION\_ACCESS in Extended CSD)*

- enum {

```

kMMC_CsdReadBlockPartialFlag = (1U << 0U),
kMMC_CsdWriteBlockMisalignFlag = (1U << 1U),
kMMC_CsdReadBlockMisalignFlag = (1U << 2U),
kMMC_CsdDsrImplementedFlag = (1U << 3U),
kMMC_CsdWriteProtectGroupEnabledFlag = (1U << 4U),
kMMC_CsdWriteBlockPartialFlag = (1U << 5U),
kMMC_ContentProtectApplicationFlag = (1U << 6U),
kMMC_CsdFileFormatGroupFlag = (1U << 7U),
kMMC_CsdCopyFlag = (1U << 8U),
kMMC_CsdPermanentWriteProtectFlag = (1U << 9U),
kMMC_CsdTemporaryWriteProtectFlag = (1U << 10U) }

```

*MMC card CSD register flags.*

- enum mmc\_extended\_csd\_access\_mode\_t {
 

```

kMMC_ExtendedCsdAccessModeCommandSet = 0U,
kMMC_ExtendedCsdAccessModeSetBits = 1U,
kMMC_ExtendedCsdAccessModeClearBits = 2U,
kMMC_ExtendedCsdAccessModeWriteBits = 3U }

```

*Extended CSD register access mode(Access mode in CMD6).*

- enum mmc\_extended\_csd\_index\_t {
 

```

kMMC_ExtendedCsdIndexFlushCache = 32U,
kMMC_ExtendedCsdIndexCacheControl = 33U,
kMMC_ExtendedCsdIndexBootPartitionWP = 173U,
kMMC_ExtendedCsdIndexEraseGroupDefinition = 175U,
kMMC_ExtendedCsdIndexBootBusConditions = 177U,
kMMC_ExtendedCsdIndexBootConfigWP = 178U,
kMMC_ExtendedCsdIndexPartitionConfig = 179U,
kMMC_ExtendedCsdIndexBusWidth = 183U,
kMMC_ExtendedCsdIndexHighSpeedTiming = 185U,
kMMC_ExtendedCsdIndexPowerClass = 187U,
kMMC_ExtendedCsdIndexCommandSet = 191U }

```

*EXT CSD byte index.*

- enum {
 

```

kMMC_DriverStrength0 = 0U,
kMMC_DriverStrength1 = 1U,
kMMC_DriverStrength2 = 2U,
kMMC_DriverStrength3 = 3U,
kMMC_DriverStrength4 = 4U }

```

*mmc driver strength*

- enum mmc\_extended\_csd\_flags\_t {
 

```

kMMC_ExtCsdExtPartitionSupport = (1 << 0U),
kMMC_ExtCsdEnhancePartitionSupport = (1 << 1U),
kMMC_ExtCsdPartitioningSupport = (1 << 2U),
kMMC_ExtCsdPrgCIDCSDInDDRModesSupport = (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` (`sdmmchost_t *host`, `uint32_t relativeAddress`, `bool isSelected`)  
*Selects the card to put it into transfer state.*
- `status_t SDMMC_SendApplicationCommand` (`sdmmchost_t *host`, `uint32_t relativeAddress`)  
*Sends an application command.*
- `status_t SDMMC_SetBlockCount` (`sdmmchost_t *host`, `uint32_t blockCount`)  
*Sets the block count.*
- `status_t SDMMC_GoIdle` (`sdmmchost_t *host`)  
*Sets the card to be idle state.*
- `status_t SDMMC_SetBlockSize` (`sdmmchost_t *host`, `uint32_t blockSize`)  
*Sets data block size.*
- `status_t SDMMC_SetCardInactive` (`sdmmchost_t *host`)  
*Sets card to inactive status.*

## 53.7.2 Data Structure Documentation

### 53.7.2.1 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*
- `sd_dat3_pull_t` `dat3PullFunc`  
*function pointer of DATA3 pull up/down*
- `void * userData`  
*user data*

### 53.7.2.2 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*

### 53.7.2.3 struct `sd_usr_param_t`

#### Data Fields

- `sd_pwr_t pwr`  
*power control configuration pointer*
- `uint32_t powerOnDelayMS`  
*power on delay time*
- `uint32_t powerOffDelayMS`  
*power off delay time*
- `sd_io_strength_t ioStrength`  
*swich 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*

### 53.7.2.4 struct `sdio_card_int_t`

#### Data Fields

- `void * userData`  
*user data*
- `sdio_int_t cardInterrupt`  
*card int call back*

### 53.7.2.5 struct `sdio_usr_param_t`

#### Data Fields

- `sd_pwr_t pwr`  
*power control configuration pointer*
- `uint32_t powerOnDelayMS`  
*power on delay time*
- `uint32_t powerOffDelayMS`  
*power off delay time*
- `sd_io_strength_t ioStrength`  
*swich 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*

### 53.7.2.6 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*

### 53.7.2.7 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*

### 53.7.2.8 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**  
*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*

### 53.7.2.9 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*

### 53.7.2.10 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].*

### 53.7.2.11 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`  
*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].*

### 53.7.2.12 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]*

### 53.7.2.13 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.*

### 53.7.2.14 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](#)  
*Manufacturer default ECC [30:29].*
- uint8\_t [writeSpeedFactor](#)  
*Write speed factor [28:26].*
- uint8\_t [maxWriteBlockLength](#)  
*Max.*
- uint8\_t [fileFormat](#)  
*File format [11:10].*
- uint8\_t [eccCode](#)  
*ECC code [9:8].*

### Field Documentation

#### (1) uint8\_t mmc\_csd\_t::transferSpeed

bus clock frequency [103:96]

#### (2) uint8\_t mmc\_csd\_t::readBlockLength

read data block length [83:80]

#### (3) uint8\_t mmc\_csd\_t::readCurrentVddMin

read current @ VDD min [61:59]

#### (4) uint8\_t mmc\_csd\_t::readCurrentVddMax

read current @ VDD max [58:56]

#### (5) uint8\_t mmc\_csd\_t::writeCurrentVddMin

write current @ VDD min [55:53]

#### (6) uint8\_t mmc\_csd\_t::writeCurrentVddMax

write current @ VDD max [52:50]

#### (7) uint8\_t mmc\_csd\_t::maxWriteBlockLength

write data block length [25:22]

## 53.7.2.15 struct mmc\_extended\_csd\_t

## Data Fields

- uint8\_t [cacheCtrl](#)  
    < secure removal type [16]
- uint8\_t [partitionAttribute](#)  
    < power off notification [34]
- uint8\_t [userWP](#)  
    < max enhance area size [159-157]
- uint8\_t [bootPartitionWP](#)  
    boot write protect register [173]
- uint8\_t [bootWPStatus](#)  
    boot write protect status register [174]
- uint8\_t [highDensityEraseGroupDefinition](#)  
    High-density erase group definition [175].
- uint8\_t [bootDataBusConditions](#)  
    Boot bus conditions [177].
- uint8\_t [bootConfigProtect](#)  
    Boot config protection [178].
- uint8\_t [partitionConfig](#)  
    Boot configuration [179].
- uint8\_t [eraseMemoryContent](#)  
    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 [partitionSwitchTimeout](#)  
    < out of interrupt busy timing [198]
- uint8\_t [powerClass52MHz195V](#)  
    Power Class for 52MHz @ 1.95V [200].
- uint8\_t [powerClass26MHz195V](#)  
    Power Class for 26MHz @ 1.95V [201].
- uint8\_t [powerClass52MHz360V](#)  
    Power Class for 52MHz @ 3.6V [202].
- uint8\_t [powerClass26MHz360V](#)

- *Power Class for 26MHz @ 3.6V [203].*
- uint8\_t **minimumReadPerformance4Bit26MHz**  
*Minimum Read Performance for 4bit at 26MHz [205].*
- uint8\_t **minimumWritePerformance4Bit26MHz**  
*Minimum Write Performance for 4bit at 26MHz [206].*
- uint8\_t **minimumReadPerformance8Bit26MHz4Bit52MHz**  
*Minimum read Performance for 8bit at 26MHz/4bit @52MHz [207].*
- uint8\_t **minimumWritePerformance8Bit26MHz4Bit52MHz**  
*Minimum Write Performance for 8bit at 26MHz/4bit @52MHz [208].*
- uint8\_t **minimumReadPerformance8Bit52MHz**  
*Minimum Read Performance for 8bit at 52MHz [209].*
- uint8\_t **minimumWritePerformance8Bit52MHz**  
*Minimum Write Performance for 8bit at 52MHz [210].*
- uint32\_t **sectorCount**  
*Sector Count [215:212].*
- uint8\_t **sleepAwakeTimeout**  
*< sleep notification timeout [216]*
- uint8\_t **sleepCurrentVCCQ**  
*< Production state awareness timeout [218]*
- uint8\_t **sleepCurrentVCC**  
*Sleep current (VCC) [220].*
- uint8\_t **highCapacityWriteProtectGroupSize**  
*High-capacity write protect group size [221].*
- uint8\_t **reliableWriteSectorCount**  
*Reliable write sector count [222].*
- uint8\_t **highCapacityEraseTimeout**  
*High-capacity erase timeout [223].*
- uint8\_t **highCapacityEraseUnitSize**  
*High-capacity erase unit size [224].*
- uint8\_t **accessSize**  
*Access size [225].*
- uint8\_t **minReadPerformance8bitAt52MHZDDR**  
*< secure trim multiplier[229]*
- uint8\_t **minWritePerformance8bitAt52MHZDDR**  
*Minimum write performance for 8bit at DDR 52MHZ[235].*
- uint8\_t **powerClass200MHZVCCQ130VVCC360V**  
*power class for 200MHZ, at VCCQ= 1.3V,VCC=3.6V[236]*
- uint8\_t **powerClass200MHZVCCQ195VVCC360V**  
*power class for 200MHZ, at VCCQ= 1.95V,VCC=3.6V[237]*
- uint8\_t **powerClass52MHZDDR195V**  
*power class for 52MHZ,DDR at Vcc 1.95V[238]*
- uint8\_t **powerClass52MHZDDR360V**  
*power class for 52MHZ,DDR at Vcc 3.6V[239]*
- uint32\_t **genericCMD6Timeout**  
*< 1st initialization time after partitioning[241]*
- uint32\_t **cacheSize**  
*cache size[252-249]*
- uint8\_t **powerClass200MHZDDR360V**  
*power class for 200MHZ, DDR at VCC=2.6V[253]*
- uint8\_t **extPartitionSupport**  
*< fw VERSION [261-254]*

- `uint8_t supportedCommandSet`  
   < *large unit size*[495]

### Field Documentation

#### (1) `uint8_t mmc_extended_csd_t::cacheCtrl`

- < product state awareness enablement[17]
- < max preload data size[21-18]
- < pre-load data size[25-22]
- < FFU status [26]
- < mode operation code[29]
- < mode config [30] control to turn on/off cache[33]

#### (2) `uint8_t mmc_extended_csd_t::partitionAttribute`

- < packed cmd fail index [35]
- < packed cmd status[36]
- < context configuration[51-37]
- < extended partitions attribut[53-52]
- < exception events status[55-54]
- < exception events control[57-56]
- < number of group to be released[58]
- < class 6 command control[59]
- < 1st initiallization after disabling sector size emu[60]
- < sector size[61]
- < sector size emulation[62]
- < native sector size[63]
- < period wakeup [131]
- < package case temperature is controlled[132]
- < production state awareness[133]
- < enhanced user data start addr [139-136]
- < enhanced user data area size[142-140]
- < general purpose partition size[154-143] partition attribute [156]

#### (3) `uint8_t mmc_extended_csd_t::userWP`

- < HPI management [161]

< write reliability parameter register[166]

< write reliability setting register[167]

< RPMB size multi [168]

< FW configuration[169] user write protect register[171]

**(4) uint8\_t mmc\_extended\_csd\_t::partitionSwitchTimeout**

partition switch timing [199]

**(5) uint8\_t mmc\_extended\_csd\_t::sleepAwakeTimeout**

Sleep/awake timeout [217]

**(6) uint8\_t mmc\_extended\_csd\_t::sleepCurrentVCCQ**

Sleep current (VCCQ) [219]

**(7) uint8\_t mmc\_extended\_csd\_t::minReadPerformance8bitAt52MHZDDR**

< secure erase multiplier[230]

< secure feature support[231]

< trim multiplier[232] Minimum read performance for 8bit at DDR 52MHZ[234]

**(8) uint32\_t mmc\_extended\_csd\_t::genericCMD6Timeout**

< correct prg sectors number[245-242]

< background operations status[246]

< power off notification timeout[247] generic CMD6 timeout[248]

**(9) uint8\_t mmc\_extended\_csd\_t::extPartitionSupport**

< device version[263-262]

< optimal trim size[264]

< optimal write size[265]

< optimal read size[266]

< pre EOL information[267]

< device life time estimation typeA[268]

< device life time estimation typeB[269]

< number of FW sectors correctly programmed[305-302]

< FFU argument[490-487]

< operation code timeout[491]



< support mode [493] extended partition attribute support[494]

#### (10) `uint8_t mmc_extended_csd_t::supportedCommandSet`

< context management capability[496]

< tag resource size[497]

< tag unit size[498]

< max packed write cmd[500]

< max packed read cmd[501]

< HPI feature[503] Supported Command Sets [504]

### 53.7.2.16 `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.*

### 53.7.2.17 `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*

### 53.7.3 Macro Definition Documentation

53.7.3.1 **#define SDMMC\_LOG( *format*, ... )**

53.7.3.2 **#define READ\_MMC\_TRANSFER\_SPEED\_FREQUENCY\_UNIT( *CSD* )((((CSD).transferSpeed) & MMC\_TRANSFER\_SPEED\_FREQUENCY\_UNIT\_MASK) >> MMC\_TRANSFER\_SPEED\_FREQUENCY\_UNIT\_SHIFT)**

53.7.3.3 **#define READ\_MMC\_TRANSFER\_SPEED\_MULTIPLIER( *CSD* )((((CSD).transferSpeed) & MMC\_TRANSFER\_SPEED\_MULTIPLIER\_MASK) >> MMC\_TRANSFER\_SPEED\_MULTIPLIER\_SHIFT)**

53.7.3.4 **#define MMC\_EXTENDED\_CSD\_BYTES (512U)**

53.7.3.5 **#define SD\_PRODUCT\_NAME\_BYTES (5U)**

53.7.3.6 **#define MMC\_PRODUCT\_NAME\_BYTES (6U)**

53.7.3.7 **#define MMC\_SWITCH\_COMMAND\_SET\_SHIFT (0U)**

53.7.3.8 **#define MMC\_SWITCH\_COMMAND\_SET\_MASK (0x0000007U)**

### 53.7.4 Enumeration Type Documentation

#### 53.7.4.1 anonymous enum

Enumerator

*kStatus\_SDMMC\_NotSupportYet* Haven't supported.  
*kStatus\_SDMMC\_TransferFailed* Send command failed.  
*kStatus\_SDMMC\_SetCardBlockSizeFailed* Set block size failed.  
*kStatus\_SDMMC\_HostNotSupport* Host doesn't support.  
*kStatus\_SDMMC\_CardNotSupport* Card doesn't support.  
*kStatus\_SDMMC\_AllSendCidFailed* Send CID failed.  
*kStatus\_SDMMC\_SendRelativeAddressFailed* Send relative address failed.  
*kStatus\_SDMMC\_SendCsdFailed* Send CSD failed.  
*kStatus\_SDMMC\_SelectCardFailed* Select card failed.  
*kStatus\_SDMMC\_SendScrFailed* Send SCR failed.  
*kStatus\_SDMMC\_SetDataBusWidthFailed* Set bus width failed.  
*kStatus\_SDMMC\_GoIdleFailed* Go idle failed.  
*kStatus\_SDMMC\_HandShakeOperationConditionFailed* Send Operation Condition failed.  
*kStatus\_SDMMC\_SendApplicationCommandFailed* Send application command failed.

*kStatus\_SDMMC\_SwitchFailed* Switch command failed.  
*kStatus\_SDMMC\_StopTransmissionFailed* Stop transmission failed.  
*kStatus\_SDMMC\_WaitWriteCompleteFailed* Wait write complete failed.  
*kStatus\_SDMMC\_SetBlockCountFailed* Set block count failed.  
*kStatus\_SDMMC\_SetRelativeAddressFailed* Set relative address failed.  
*kStatus\_SDMMC\_SwitchBusTimingFailed* Switch high speed failed.  
*kStatus\_SDMMC\_SendExtendedCsdFailed* Send EXT\_CSD failed.  
*kStatus\_SDMMC\_ConfigureBootFailed* Configure boot failed.  
*kStatus\_SDMMC\_ConfigureExtendedCsdFailed* Configure EXT\_CSD failed.  
*kStatus\_SDMMC\_EnableHighCapacityEraseFailed* Enable high capacity erase failed.  
*kStatus\_SDMMC\_SendTestPatternFailed* Send test pattern failed.  
*kStatus\_SDMMC\_ReceiveTestPatternFailed* Receive test pattern failed.  
*kStatus\_SDMMC\_SDIO\_ResponseError* sdio response error  
*kStatus\_SDMMC\_SDIO\_InvalidArgument* sdio invalid argument response error  
*kStatus\_SDMMC\_SDIO\_SendOperationConditionFail* sdio send operation condition fail  
*kStatus\_SDMMC\_InvalidVoltage* invalid voltage  
*kStatus\_SDMMC\_SDIO\_SwitchHighSpeedFail* switch to high speed fail  
*kStatus\_SDMMC\_SDIO\_ReadCISFail* read CIS fail  
*kStatus\_SDMMC\_SDIO\_InvalidCard* invalid SDIO card  
*kStatus\_SDMMC\_TuningFail* tuning fail  
*kStatus\_SDMMC\_SwitchVoltageFail* switch voltage fail  
*kStatus\_SDMMC\_SwitchVoltage18VFail33VSuccess* switch voltage fail  
*kStatus\_SDMMC\_ReTuningRequest* retuning request  
*kStatus\_SDMMC\_SetDriverStrengthFail* set driver strength fail  
*kStatus\_SDMMC\_SetPowerClassFail* set power class fail  
*kStatus\_SDMMC\_HostNotReady* host controller not ready  
*kStatus\_SDMMC\_CardDetectFailed* card detect failed  
*kStatus\_SDMMC\_AuSizeNotSetProperly* AU size not set properly.  
*kStatus\_SDMMC\_PollingCardIdleFailed* polling card idle status failed  
*kStatus\_SDMMC\_DeselectCardFailed* deselect card failed  
*kStatus\_SDMMC\_CardStatusIdle* card idle  
*kStatus\_SDMMC\_CardStatusBusy* card busy  
*kStatus\_SDMMC\_CardInitFailed* card init failed

#### 53.7.4.2 anonymous enum

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

#### 53.7.4.3 enum sdmmc\_operation\_voltage\_t

Enumerator

*kSDMMC\_OperationVoltageNone* indicate current voltage setting is not setting by suser  
*kSDMMC\_OperationVoltage330V* card operation voltage around 3.3v  
*kSDMMC\_OperationVoltage300V* card operation voltage around 3.0v  
*kSDMMC\_OperationVoltage180V* card operation voltage around 1.8v

#### 53.7.4.4 anonymous enum

Enumerator

*kSDMMC\_BusWdith1Bit* card bus 1 width  
*kSDMMC\_BusWdith4Bit* card bus 4 width  
*kSDMMC\_BusWdith8Bit* card bus 8 width

#### 53.7.4.5 anonymous enum

Enumerator

*kSDMMC\_Support8BitWidth* 8 bit data width capability

#### 53.7.4.6 anonymous enum

Enumerator

*kSDMMC\_DataPacketFormatLSBFirst* usual data packet format LSB first, MSB last  
*kSDMMC\_DataPacketFormatMSBFirst* Wide width data packet format MSB first, LSB last.

#### 53.7.4.7 enum sd\_detect\_card\_type\_t

Enumerator

*kSD\_DetectCardByGpioCD* sd card detect by CD pin through GPIO  
*kSD\_DetectCardByHostCD* sd card detect by CD pin through host  
*kSD\_DetectCardByHostDATA3* sd card detect by DAT3 pin through host

**53.7.4.8 anonymous enum**

Enumerator

*kSD\_Inserted* card is inserted  
*kSD\_Removed* card is removed

**53.7.4.9 anonymous enum**

Enumerator

*kSD\_DAT3PullDown* data3 pull down  
*kSD\_DAT3PullUp* data3 pull up

**53.7.4.10 enum sd\_io\_voltage\_ctrl\_type\_t**

Enumerator

*kSD\_IOVoltageCtrlNotSupport* io voltage control not support  
*kSD\_IOVoltageCtrlByGpio* io voltage control by gpio

**53.7.4.11 anonymous enum**

Enumerator

*kSDMMC\_R1OutOfRangeFlag* Out of range status bit.  
*kSDMMC\_R1AddressErrorFlag* Address error status bit.  
*kSDMMC\_R1BlockLengthErrorFlag* Block length error status bit.  
*kSDMMC\_R1EraseSequenceErrorFlag* Erase sequence error status bit.  
*kSDMMC\_R1EraseParameterErrorFlag* Erase parameter error status bit.  
*kSDMMC\_R1WriteProtectViolationFlag* Write protection violation status bit.  
*kSDMMC\_R1CardIsLockedFlag* Card locked status bit.  
*kSDMMC\_R1LockUnlockFailedFlag* lock/unlock error status bit  
*kSDMMC\_R1CommandCrcErrorFlag* CRC error status bit.  
*kSDMMC\_R1IllegalCommandFlag* Illegal command status bit.  
*kSDMMC\_R1CardEccFailedFlag* Card ecc error status bit.  
*kSDMMC\_R1CardControllerErrorFlag* Internal card controller error status bit.  
*kSDMMC\_R1ErrorFlag* A general or an unknown error status bit.  
*kSDMMC\_R1CidCsdOverwriteFlag* Cid/csd overwrite status bit.  
*kSDMMC\_R1WriteProtectEraseSkipFlag* Write protection erase skip status bit.  
*kSDMMC\_R1CardEccDisabledFlag* Card ecc disabled status bit.  
*kSDMMC\_R1EraseResetFlag* Erase reset status bit.  
*kSDMMC\_R1ReadyForDataFlag* Ready for data status bit.  
*kSDMMC\_R1SwitchErrorFlag* Switch error status bit.  
*kSDMMC\_R1ApplicationCommandFlag* Application command enabled status bit.

***kSDMMC\_R1AuthenticationSequenceErrorFlag*** error in the sequence of authentication process

#### 53.7.4.12 enum sdmmc\_r1\_current\_state\_t

Enumerator

***kSDMMC\_R1StateIdle*** R1: current state: idle.  
***kSDMMC\_R1StateReady*** R1: current state: ready.  
***kSDMMC\_R1StateIdentify*** R1: current state: identification.  
***kSDMMC\_R1StateStandby*** R1: current state: standby.  
***kSDMMC\_R1StateTransfer*** R1: current state: transfer.  
***kSDMMC\_R1StateSendData*** R1: current state: sending data.  
***kSDMMC\_R1StateReceiveData*** R1: current state: receiving data.  
***kSDMMC\_R1StateProgram*** R1: current state: programming.  
***kSDMMC\_R1StateDisconnect*** R1: current state: disconnect.

#### 53.7.4.13 anonymous enum

Enumerator

***kSDSPI\_R1InIdleStateFlag*** In idle state.  
***kSDSPI\_R1EraseResetFlag*** Erase reset.  
***kSDSPI\_R1IllegalCommandFlag*** Illegal command.  
***kSDSPI\_R1CommandCrcErrorFlag*** Com crc error.  
***kSDSPI\_R1EraseSequenceErrorFlag*** Erase sequence error.  
***kSDSPI\_R1AddressErrorFlag*** Address error.  
***kSDSPI\_R1ParameterErrorFlag*** Parameter error.

#### 53.7.4.14 anonymous enum

Enumerator

***kSDSPI\_R2CardLockedFlag*** Card is locked.  
***kSDSPI\_R2WriteProtectEraseSkip*** Write protect erase skip.  
***kSDSPI\_R2LockUnlockFailed*** Lock/unlock command failed.  
***kSDSPI\_R2ErrorFlag*** Unknown error.  
***kSDSPI\_R2CardControllerErrorFlag*** Card controller error.  
***kSDSPI\_R2CardEccFailedFlag*** Card ecc failed.  
***kSDSPI\_R2WriteProtectViolationFlag*** Write protect violation.  
***kSDSPI\_R2EraseParameterErrorFlag*** Erase parameter error.  
***kSDSPI\_R2OutOfRangeFlag*** Out of range.  
***kSDSPI\_R2CsdOverwriteFlag*** CSD overwrite.

**53.7.4.15 anonymous enum**

Enumerator

*kSDSPI\_DataErrorTokenError* Data error.  
*kSDSPI\_DataErrorTokenCardControllerError* Card controller error.  
*kSDSPI\_DataErrorTokenCardEccFailed* Card ecc error.  
*kSDSPI\_DataErrorTokenOutOfRange* Out of range.

**53.7.4.16 enum sdspi\_data\_token\_t**

Enumerator

*kSDSPI\_DataTokenBlockRead* Single block read, multiple block read.  
*kSDSPI\_DataTokenSingleBlockWrite* Single block write.  
*kSDSPI\_DataTokenMultipleBlockWrite* Multiple block write.  
*kSDSPI\_DataTokenStopTransfer* Stop transmission.

**53.7.4.17 enum sdspi\_data\_response\_token\_t**

Enumerator

*kSDSPI\_DataResponseTokenAccepted* Data accepted.  
*kSDSPI\_DataResponseTokenCrcError* Data rejected due to CRC error.  
*kSDSPI\_DataResponseTokenWriteError* Data rejected due to write error.

**53.7.4.18 enum sd\_command\_t**

Enumerator

*kSD\_SendRelativeAddress* Send Relative Address.  
*kSD\_Switch* Switch Function.  
*kSD\_SendInterfaceCondition* Send Interface Condition.  
*kSD\_VoltageSwitch* Voltage Switch.  
*kSD\_SpeedClassControl* Speed Class control.  
*kSD\_EraseWriteBlockStart* Write Block Start.  
*kSD\_EraseWriteBlockEnd* Write Block End.  
*kSD\_SendTuningBlock* Send Tuning Block.

**53.7.4.19 enum sdspi\_command\_t**

Enumerator

*kSDSPI\_CommandCrc* Command crc protection on/off.

**53.7.4.20 enum sd\_application\_command\_t**

Enumerator

*kSD\_ApplicationSetBusWdith* Set Bus Width.  
*kSD\_ApplicationStatus* Send SD status.  
*kSD\_ApplicationSendNumberWriteBlocks* Send Number Of Written Blocks.  
*kSD\_ApplicationSetWriteBlockEraseCount* Set Write Block Erase Count.  
*kSD\_ApplicationSendOperationCondition* Send Operation Condition.  
*kSD\_ApplicationSetClearCardDetect* Set Connect/Disconnect pull up on detect pin.  
*kSD\_ApplicationSendScr* Send Scr.

**53.7.4.21 anonymous enum**

Enumerator

*kSDMMC\_CommandClassBasic* Card command class 0.  
*kSDMMC\_CommandClassBlockRead* Card command class 2.  
*kSDMMC\_CommandClassBlockWrite* Card command class 4.  
*kSDMMC\_CommandClassErase* Card command class 5.  
*kSDMMC\_CommandClassWriteProtect* Card command class 6.  
*kSDMMC\_CommandClassLockCard* Card command class 7.  
*kSDMMC\_CommandClassApplicationSpecific* Card command class 8.  
*kSDMMC\_CommandClassInputOutputMode* Card command class 9.  
*kSDMMC\_CommandClassSwitch* Card command class 10.

**53.7.4.22 anonymous enum**

Enumerator

*kSD\_OcrPowerUpBusyFlag* Power up busy status.  
*kSD\_OcrHostCapacitySupportFlag* Card capacity status.  
*kSD\_OcrCardCapacitySupportFlag* Card capacity status.  
*kSD\_OcrSwitch18RequestFlag* Switch to 1.8V request.  
*kSD\_OcrSwitch18AcceptFlag* Switch to 1.8V accepted.  
*kSD\_OcrVdd27\_28Flag* VDD 2.7-2.8.  
*kSD\_OcrVdd28\_29Flag* VDD 2.8-2.9.  
*kSD\_OcrVdd29\_30Flag* VDD 2.9-3.0.  
*kSD\_OcrVdd30\_31Flag* VDD 2.9-3.0.  
*kSD\_OcrVdd31\_32Flag* VDD 3.0-3.1.  
*kSD\_OcrVdd32\_33Flag* VDD 3.1-3.2.  
*kSD\_OcrVdd33\_34Flag* VDD 3.2-3.3.  
*kSD\_OcrVdd34\_35Flag* VDD 3.3-3.4.  
*kSD\_OcrVdd35\_36Flag* VDD 3.4-3.5.



**53.7.4.23 anonymous enum**

Enumerator

*kSD\_SpecificationVersion1\_0* SD card version 1.0-1.01.  
*kSD\_SpecificationVersion1\_1* SD card version 1.10.  
*kSD\_SpecificationVersion2\_0* SD card version 2.00.  
*kSD\_SpecificationVersion3\_0* SD card version 3.0.

**53.7.4.24 enum sd\_switch\_mode\_t**

Enumerator

*kSD\_SwitchCheck* SD switch mode 0: check function.  
*kSD\_SwitchSet* SD switch mode 1: set function.

**53.7.4.25 anonymous enum**

Enumerator

*kSD\_CsdReadBlockPartialFlag* Partial blocks for read allowed [79:79].  
*kSD\_CsdWriteBlockMisalignFlag* Write block misalignment [78:78].  
*kSD\_CsdReadBlockMisalignFlag* Read block misalignment [77:77].  
*kSD\_CsdDsrImplementedFlag* DSR implemented [76:76].  
*kSD\_CsdEraseBlockEnabledFlag* Erase single block enabled [46:46].  
*kSD\_CsdWriteProtectGroupEnabledFlag* Write protect group enabled [31:31].  
*kSD\_CsdWriteBlockPartialFlag* Partial blocks for write allowed [21:21].  
*kSD\_CsdFileFormatGroupFlag* File format group [15:15].  
*kSD\_CsdCopyFlag* Copy flag [14:14].  
*kSD\_CsdPermanentWriteProtectFlag* Permanent write protection [13:13].  
*kSD\_CsdTemporaryWriteProtectFlag* Temporary write protection [12:12].

**53.7.4.26 anonymous enum**

Enumerator

*kSD\_ScrDataStatusAfterErase* Data status after erases [55:55].  
*kSD\_ScrSdSpecification3* Specification version 3.00 or higher [47:47].

**53.7.4.27 anonymous enum**

Enumerator

*kSD\_FunctionSDR12Deafult* SDR12 mode & default.  
*kSD\_FunctionSDR25HighSpeed* SDR25 & high speed.

*kSD\_FunctionSDR50* SDR50 mode.  
*kSD\_FunctionSDR104* SDR104 mode.  
*kSD\_FunctionDDR50* DDR50 mode.

#### 53.7.4.28 anonymous enum

Enumerator

*kSD\_GroupTimingMode* access mode group  
*kSD\_GroupCommandSystem* command system group  
*kSD\_GroupDriverStrength* driver strength group  
*kSD\_GroupCurrentLimit* current limit group

#### 53.7.4.29 enum sd\_timing\_mode\_t

Enumerator

*kSD\_TimingSDR12DefaultMode* Identification mode & SDR12.  
*kSD\_TimingSDR25HighSpeedMode* High speed mode & SDR25.  
*kSD\_TimingSDR50Mode* SDR50 mode.  
*kSD\_TimingSDR104Mode* SDR104 mode.  
*kSD\_TimingDDR50Mode* DDR50 mode.

#### 53.7.4.30 enum sd\_driver\_strength\_t

Enumerator

*kSD\_DriverStrengthTypeB* default driver strength  
*kSD\_DriverStrengthTypeA* driver strength TYPE A  
*kSD\_DriverStrengthTypeC* driver strength TYPE C  
*kSD\_DriverStrengthTypeD* driver strength TYPE D

#### 53.7.4.31 enum sd\_max\_current\_t

Enumerator

*kSD\_CurrentLimit200MA* default current limit  
*kSD\_CurrentLimit400MA* current limit to 400MA  
*kSD\_CurrentLimit600MA* current limit to 600MA  
*kSD\_CurrentLimit800MA* current limit to 800MA

**53.7.4.32 enum sdmmc\_command\_t**

Enumerator

*kSDMMC\_GoIdleState* Go Idle State.  
*kSDMMC\_AllSendCid* All Send CID.  
*kSDMMC\_SetDsr* Set DSR.  
*kSDMMC\_SelectCard* Select Card.  
*kSDMMC\_SendCsd* Send CSD.  
*kSDMMC\_SendCid* Send CID.  
*kSDMMC\_StopTransmission* Stop Transmission.  
*kSDMMC\_SendStatus* Send Status.  
*kSDMMC\_GoInactiveState* Go Inactive State.  
*kSDMMC\_SetBlockLength* Set Block Length.  
*kSDMMC\_ReadSingleBlock* Read Single Block.  
*kSDMMC\_ReadMultipleBlock* Read Multiple Block.  
*kSDMMC\_SetBlockCount* Set Block Count.  
*kSDMMC\_WriteSingleBlock* Write Single Block.  
*kSDMMC\_WriteMultipleBlock* Write Multiple Block.  
*kSDMMC\_ProgramCsd* Program CSD.  
*kSDMMC\_SetWriteProtect* Set Write Protect.  
*kSDMMC\_ClearWriteProtect* Clear Write Protect.  
*kSDMMC\_SendWriteProtect* Send Write Protect.  
*kSDMMC\_Erase* Erase.  
*kSDMMC\_LockUnlock* Lock Unlock.  
*kSDMMC\_ApplicationCommand* Send Application Command.  
*kSDMMC\_GeneralCommand* General Purpose Command.  
*kSDMMC\_ReadOcr* Read OCR.

**53.7.4.33 anonymous enum**

Enumerator

*kSDIO\_RegCCCRSdioVer* CCCR & SDIO version.  
*kSDIO\_RegSDVersion* SD version.  
*kSDIO\_RegIOEnable* io enable register  
*kSDIO\_RegIOReady* io ready register  
*kSDIO\_RegIOIntEnable* io interrupt enable register  
*kSDIO\_RegIOIntPending* io interrupt pending register  
*kSDIO\_RegIOAbort* io abort register  
*kSDIO\_RegBusInterface* bus interface register  
*kSDIO\_RegCardCapability* card capability register  
*kSDIO\_RegCommonCISPointer* common CIS pointer register  
*kSDIO\_RegBusSuspend* bus suspend register  
*kSDIO\_RegFunctionSelect* function select register  
*kSDIO\_RegExecutionFlag* execution flag register

*kSDIO\_RegReadyFlag* ready flag register  
*kSDIO\_RegFN0BlockSizeLow* FN0 block size register.  
*kSDIO\_RegFN0BlockSizeHigh* FN0 block size register.  
*kSDIO\_RegPowerControl* power control register  
*kSDIO\_RegBusSpeed* bus speed register  
*kSDIO\_RegUHSITimingSupport* UHS-I timing support register.  
*kSDIO\_RegDriverStrength* Driver strength register.  
*kSDIO\_RegInterruptExtension* Interrupt extension register.

#### 53.7.4.34 enum sdio\_command\_t

Enumerator

*kSDIO\_SendRelativeAddress* send relative address  
*kSDIO\_SendOperationCondition* send operation condition  
*kSDIO\_SendInterfaceCondition* send interface condition  
*kSDIO\_RWIODirect* read/write IO direct command  
*kSDIO\_RWIOExtended* read/write IO extended command

#### 53.7.4.35 enum sdio\_func\_num\_t

Enumerator

*kSDIO\_FunctionNum0* sdio function0  
*kSDIO\_FunctionNum1* sdio function1  
*kSDIO\_FunctionNum2* sdio function2  
*kSDIO\_FunctionNum3* sdio function3  
*kSDIO\_FunctionNum4* sdio function4  
*kSDIO\_FunctionNum5* sdio function5  
*kSDIO\_FunctionNum6* sdio function6  
*kSDIO\_FunctionNum7* sdio function7  
*kSDIO\_FunctionMemory* for combo card

#### 53.7.4.36 anonymous enum

Enumerator

*kSDIO\_StatusCmdCRCError* the CRC check of the previous cmd fail  
*kSDIO\_StatusIllegalCmd* cmd illegal for the card state  
*kSDIO\_StatusR6Error* special for R6 error status  
*kSDIO\_StatusError* A general or an unknown error occurred.  
*kSDIO\_StatusFunctionNumError* inval function error  
*kSDIO\_StatusOutOfRange* cmd argument was out of the allowed range

**53.7.4.37 anonymous enum**

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.

**53.7.4.38 anonymous enum**

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

**53.7.4.39 anonymous enum**

Enumerator

*kSDIO\_FBRSupportCSA* function support CSA  
*kSDIO\_FBRSupportPowerSelection* function support power selection

**53.7.4.40 enum sdio\_bus\_width\_t**

Enumerator

*kSDIO\_DataBus1Bit* 1 bit bus mode  
*kSDIO\_DataBus4Bit* 4 bit bus mode  
*kSDIO\_DataBus8Bit* 8 bit bus mode

**53.7.4.41 enum mmc\_command\_t**

Enumerator

*kMMC\_SendOperationCondition* Send Operation Condition.  
*kMMC\_SetRelativeAddress* Set Relative Address.  
*kMMC\_SleepAwake* Sleep Awake.  
*kMMC\_Switch* Switch.  
*kMMC\_SendExtendedCsd* Send EXT\_CSD.  
*kMMC\_ReadDataUntilStop* Read Data Until Stop.  
*kMMC\_BusTestRead* Test Read.  
*kMMC\_SendingBusTest* test bus width cmd  
*kMMC\_WriteDataUntilStop* Write Data Until Stop.  
*kMMC\_SendTuningBlock* MMC sending tuning block.  
*kMMC\_ProgramCid* Program CID.  
*kMMC\_EraseGroupStart* Erase Group Start.  
*kMMC\_EraseGroupEnd* Erase Group End.  
*kMMC\_FastInputOutput* Fast IO.  
*kMMC\_GoInterruptState* Go interrupt State.

**53.7.4.42 enum mmc\_classified\_voltage\_t**

Enumerator

*kMMC\_ClassifiedVoltageHigh* High-voltage MMC card.  
*kMMC\_ClassifiedVoltageDual* Dual-voltage MMC card.

**53.7.4.43 enum mmc\_classified\_density\_t**

Enumerator

*kMMC\_ClassifiedDensityWithin2GB* Density byte is less than or equal 2GB.

**53.7.4.44 enum mmc\_access\_mode\_t**

Enumerator

- kMMC\_AccessModeByte* The card should be accessed as byte.
- kMMC\_AccessModeSector* The card should be accessed as sector.

**53.7.4.45 enum mmc\_voltage\_window\_t**

Enumerator

- kMMC\_VoltageWindowNone* voltage window is not define by user
- kMMC\_VoltageWindow120* Voltage window is 1.20V.
- kMMC\_VoltageWindow170to195* Voltage window is 1.70V to 1.95V.
- kMMC\_VoltageWindows270to360* Voltage window is 2.70V to 3.60V.

**53.7.4.46 enum mmc\_csd\_structure\_version\_t**

Enumerator

- kMMC\_CsdStrucureVersion10* CSD version No. 1.0
- kMMC\_CsdStrucureVersion11* CSD version No. 1.1
- kMMC\_CsdStrucureVersion12* CSD version No. 1.2
- kMMC\_CsdStrucureVersionInExtcsd* Version coded in Extended CSD.

**53.7.4.47 enum mmc\_specification\_version\_t**

Enumerator

- kMMC\_SpecificationVersion0* Allocated by MMCA.
- kMMC\_SpecificationVersion1* Allocated by MMCA.
- kMMC\_SpecificationVersion2* Allocated by MMCA.
- kMMC\_SpecificationVersion3* Allocated by MMCA.
- kMMC\_SpecificationVersion4* Version 4.1/4.2/4.3/4.41-4.5-4.51-5.0.

**53.7.4.48 anonymous enum**

Enumerator

- kMMC\_ExtendedCsdRevision10* Revision 1.0.
- kMMC\_ExtendedCsdRevision11* Revision 1.1.
- kMMC\_ExtendedCsdRevision12* Revision 1.2.
- kMMC\_ExtendedCsdRevision13* Revision 1.3 MMC4.3.
- kMMC\_ExtendedCsdRevision14* Revision 1.4 obsolete.

*kMMC\_ExtendedCsdRevision15* Revision 1.5 MMC4.41.  
*kMMC\_ExtendedCsdRevision16* Revision 1.6 MMC4.5.  
*kMMC\_ExtendedCsdRevision17* Revision 1.7 MMC5.0.

#### 53.7.4.49 enum mmc\_command\_set\_t

Enumerator

*kMMC\_CommandSetStandard* Standard MMC.  
*kMMC\_CommandSet1* Command set 1.  
*kMMC\_CommandSet2* Command set 2.  
*kMMC\_CommandSet3* Command set 3.  
*kMMC\_CommandSet4* Command set 4.

#### 53.7.4.50 anonymous enum

Enumerator

*kMMC\_SupportAlternateBoot* support alternative boot mode  
*kMMC\_SupportDDRBoot* support DDR boot mode  
*kMMC\_SupportHighSpeedBoot* support high speed boot mode

#### 53.7.4.51 enum mmc\_high\_speed\_timing\_t

Enumerator

*kMMC\_HighSpeedTimingNone* MMC card using none high-speed timing.  
*kMMC\_HighSpeedTiming* MMC card using high-speed timing.  
*kMMC\_HighSpeed200Timing* MMC card high speed 200 timing.  
*kMMC\_HighSpeed400Timing* MMC card high speed 400 timing.  
*kMMC\_EnhanceHighSpeed400Timing* MMC card high speed 400 timing.

#### 53.7.4.52 enum mmc\_data\_bus\_width\_t

Enumerator

*kMMC\_DataBusWidth1bit* MMC data bus width is 1 bit.  
*kMMC\_DataBusWidth4bit* MMC data bus width is 4 bits.  
*kMMC\_DataBusWidth8bit* MMC data bus width is 8 bits.  
*kMMC\_DataBusWidth4bitDDR* MMC data bus width is 4 bits ddr.  
*kMMC\_DataBusWidth8bitDDR* MMC data bus width is 8 bits ddr.  
*kMMC\_DataBusWidth8bitDDRSTROBE* MMC data bus width is 8 bits ddr strobe mode.



**53.7.4.53 enum mmc\_boot\_partition\_enable\_t**

Enumerator

- kMMC\_BootPartitionEnableNot* Device not boot enabled (default)
- kMMC\_BootPartitionEnablePartition1* Boot partition 1 enabled for boot.
- kMMC\_BootPartitionEnablePartition2* Boot partition 2 enabled for boot.
- kMMC\_BootPartitionEnableUserAera* User area enabled for boot.

**53.7.4.54 enum mmc\_boot\_timing\_mode\_t**

Enumerator

- kMMC\_BootModeSDRWithDefaultTiming* boot mode single data rate with backward compatible timings
- kMMC\_BootModeSDRWithHighSpeedTiming* boot mode single data rate with high speed timing
- kMMC\_BootModeDDRTiming* boot mode dual data rate

**53.7.4.55 enum mmc\_boot\_partition\_wp\_t**

Enumerator

- kMMC\_BootPartitionWPDisable* boot partition write protection disable
- kMMC\_BootPartitionPwrWPToBothPartition* power on period write protection apply to both boot partitions
- kMMC\_BootPartitionPermWPToBothPartition* permanent write protection apply to both boot partitions
- kMMC\_BootPartitionPwrWPToPartition1* power on period write protection apply to partition1
- kMMC\_BootPartitionPwrWPToPartition2* power on period write protection apply to partition2
- kMMC\_BootPartitionPermWPToPartition1* permanent write protection apply to partition1
- kMMC\_BootPartitionPermWPToPartition2* permanent write protection apply to partition2
- kMMC\_BootPartitionPermWPToPartition1PwrWPToPartition2* permanent write protection apply to partition1, power on period write protection apply to partition2
- kMMC\_BootPartitionPermWPToPartition2PwrWPToPartition1* permanent write protection apply to partition2, power on period write protection apply to partition1

**53.7.4.56 anonymous enum**

Enumerator

- kMMC\_BootPartitionNotProtected* boot partition not protected
- kMMC\_BootPartitionPwrProtected* boot partition is power on period write protected
- kMMC\_BootPartitionPermProtected* boot partition is permanently protected

**53.7.4.57 enum mmc\_access\_partition\_t**

Enumerator

*kMMC\_AccessPartitionUserAera* No access to boot partition (default), normal partition.  
*kMMC\_AccessPartitionBoot1* Read/Write boot partition 1.  
*kMMC\_AccessPartitionBoot2* Read/Write boot partition 2.  
*kMMC\_AccessRPMB* Replay protected mem block.  
*kMMC\_AccessGeneralPurposePartition1* access to general purpose partition 1  
*kMMC\_AccessGeneralPurposePartition2* access to general purpose partition 2  
*kMMC\_AccessGeneralPurposePartition3* access to general purpose partition 3  
*kMMC\_AccessGeneralPurposePartition4* access to general purpose partition 4

**53.7.4.58 anonymous enum**

Enumerator

*kMMC\_CsdReadBlockPartialFlag* Partial blocks for read allowed.  
*kMMC\_CsdWriteBlockMisalignFlag* Write block misalignment.  
*kMMC\_CsdReadBlockMisalignFlag* Read block misalignment.  
*kMMC\_CsdDsrImplementedFlag* DSR implemented.  
*kMMC\_CsdWriteProtectGroupEnabledFlag* Write protect group enabled.  
*kMMC\_CsdWriteBlockPartialFlag* Partial blocks for write allowed.  
*kMMC\_ContentProtectApplicationFlag* Content protect application.  
*kMMC\_CsdFileFormatGroupFlag* File format group.  
*kMMC\_CsdCopyFlag* Copy flag.  
*kMMC\_CsdPermanentWriteProtectFlag* Permanent write protection.  
*kMMC\_CsdTemporaryWriteProtectFlag* Temporary write protection.

**53.7.4.59 enum mmc\_extended\_csd\_access\_mode\_t**

Enumerator

*kMMC\_ExtendedCsdAccessModeCommandSet* Command set related setting.  
*kMMC\_ExtendedCsdAccessModeSetBits* Set bits in specific byte in Extended CSD.  
*kMMC\_ExtendedCsdAccessModeClearBits* Clear bits in specific byte in Extended CSD.  
*kMMC\_ExtendedCsdAccessModeWriteBits* Write a value to specific byte in Extended CSD.

**53.7.4.60 enum mmc\_extended\_csd\_index\_t**

Enumerator

*kMMC\_ExtendedCsdIndexFlushCache* flush cache  
*kMMC\_ExtendedCsdIndexCacheControl* cache control

***kMMC\_ExtendedCsdIndexBootPartitionWP*** Boot partition write protect.  
***kMMC\_ExtendedCsdIndexEraseGroupDefinition*** Erase Group Def.  
***kMMC\_ExtendedCsdIndexBootBusConditions*** Boot Bus conditions.  
***kMMC\_ExtendedCsdIndexBootConfigWP*** Boot config write protect.  
***kMMC\_ExtendedCsdIndexPartitionConfig*** Partition Config, before BOOT\_CONFIG.  
***kMMC\_ExtendedCsdIndexBusWidth*** Bus Width.  
***kMMC\_ExtendedCsdIndexHighSpeedTiming*** High-speed Timing.  
***kMMC\_ExtendedCsdIndexPowerClass*** Power Class.  
***kMMC\_ExtendedCsdIndexCommandSet*** Command Set.

#### 53.7.4.61 anonymous enum

Enumerator

***kMMC\_DriverStrength0*** Driver type0 ,nominal impedance 50ohm.  
***kMMC\_DriverStrength1*** Driver type1 ,nominal impedance 33ohm.  
***kMMC\_DriverStrength2*** Driver type2 ,nominal impedance 66ohm.  
***kMMC\_DriverStrength3*** Driver type3 ,nominal impedance 100ohm.  
***kMMC\_DriverStrength4*** Driver type4 ,nominal impedance 40ohm.

#### 53.7.4.62 enum mmc\_extended\_csd\_flags\_t

Enumerator

***kMMC\_ExtCsdExtPartitionSupport*** partitioning support[160]  
***kMMC\_ExtCsdEnhancePartitionSupport*** partitioning support[160]  
***kMMC\_ExtCsdPartitioningSupport*** partitioning support[160]  
***kMMC\_ExtCsdPrgCIDCSDInDDRModesSupport*** 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]

#### 53.7.4.63 enum mmc\_boot\_mode\_t

Enumerator

***kMMC\_BootModeNormal*** Normal boot.  
***kMMC\_BootModeAlternative*** Alternative boot.

## 53.7.5 Function Documentation

53.7.5.1 `status_t SDMMC_SelectCard ( sdmmc_host_t * host, uint32_t relativeAddress, bool isSelected )`

## Parameters

|                        |                                       |
|------------------------|---------------------------------------|
| <i>host</i>            | host handler.                         |
| <i>relativeAddress</i> | Relative address.                     |
| <i>isSelected</i>      | True to put card into transfer state. |

## Return values

|                                      |                       |
|--------------------------------------|-----------------------|
| <i>kStatus_SDMMC_-TransferFailed</i> | Transfer failed.      |
| <i>kStatus_Success</i>               | Operate successfully. |

### 53.7.5.2 **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. |

### 53.7.5.3 **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. |

**53.7.5.4 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. |

**53.7.5.5 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. |

**53.7.5.6 status\_t SDMMC\_SetCardInactive ( sdmmc\_host\_t \* host )**

Parameters

\_\_\_\_\_

|             |               |
|-------------|---------------|
| <i>host</i> | host handler. |
|-------------|---------------|

Return values

|                                           |                       |
|-------------------------------------------|-----------------------|
| <i>kStatus_SDMMC_-<br/>TransferFailed</i> | Transfer failed.      |
| <i>kStatus_Success</i>                    | Operate successfully. |



# Chapter 54

## CODEC Driver

### 54.1 Overview

The MCUXpresso SDK provides a codec abstraction driver interface to access codec register.

#### Modules

- [CODEC Common Driver](#)
- [CODEC I2C Driver](#)
- [CS42888 Driver](#)
- [DA7212 Driver](#)
- [SGTL5000 Driver](#)
- [WM8904 Driver](#)
- [WM8960 Driver](#)



## 54.2 CODEC Common Driver

### 54.2.1 Overview

The codec common driver provides a codec control abstraction interface.

#### Modules

- [CODEC Adapter](#)
- [CS42888 Adapter](#)
- [DA7212 Adapter](#)
- [SGTL5000 Adapter](#)
- [WM8904 Adapter](#)
- [WM8960 Adapter](#)

#### Data Structures

- struct [codec\\_config\\_t](#)  
*Initialize structure of the codec. [More...](#)*
- struct [codec\\_capability\\_t](#)  
*codec capability [More...](#)*
- struct [codec\\_handle\\_t](#)  
*Codec handle definition. [More...](#)*

#### Macros

- `#define CODEC\_VOLUME\_MAX\_VALUE (100U)`  
*codec maximum volume range*

#### Enumerations

- enum {  
  [kStatus\\_CODEC\\_NotSupport](#) = MAKE\_STATUS(kStatusGroup\_CODEC, 0U),  
  [kStatus\\_CODEC\\_DeviceNotRegistered](#) = MAKE\_STATUS(kStatusGroup\_CODEC, 1U),  
  [kStatus\\_CODEC\\_I2CBusInitialFailed](#),  
  [kStatus\\_CODEC\\_I2CCommandTransferFailed](#) }  
*CODEC status.*
- enum [codec\\_audio\\_protocol\\_t](#) {  
  [kCODEC\\_BusI2S](#) = 0U,  
  [kCODEC\\_BusLeftJustified](#) = 1U,  
  [kCODEC\\_BusRightJustified](#) = 2U,  
  [kCODEC\\_BusPCMA](#) = 3U,  
  [kCODEC\\_BusPCMB](#) = 4U,  
  [kCODEC\\_BusTDM](#) = 5U }

- AUDIO format definition.*
  - enum {
    - kCODEC\_AudioSampleRate8KHz = 8000U,
    - kCODEC\_AudioSampleRate11025Hz = 11025U,
    - kCODEC\_AudioSampleRate12KHz = 12000U,
    - kCODEC\_AudioSampleRate16KHz = 16000U,
    - kCODEC\_AudioSampleRate22050Hz = 22050U,
    - kCODEC\_AudioSampleRate24KHz = 24000U,
    - kCODEC\_AudioSampleRate32KHz = 32000U,
    - kCODEC\_AudioSampleRate44100Hz = 44100U,
    - kCODEC\_AudioSampleRate48KHz = 48000U,
    - kCODEC\_AudioSampleRate96KHz = 96000U,
    - kCODEC\_AudioSampleRate192KHz = 192000U,
    - kCODEC\_AudioSampleRate384KHz = 384000U }
  - audio sample rate definition*
    - enum {
      - kCODEC\_AudioBitWidth16bit = 16U,
      - kCODEC\_AudioBitWidth20bit = 20U,
      - kCODEC\_AudioBitWidth24bit = 24U,
      - kCODEC\_AudioBitWidth32bit = 32U }
    - audio bit width*
      - enum codec\_module\_t {
        - kCODEC\_ModuleADC = 0U,
        - kCODEC\_ModuleDAC = 1U,
        - kCODEC\_ModulePGA = 2U,
        - kCODEC\_ModuleHeadphone = 3U,
        - kCODEC\_ModuleSpeaker = 4U,
        - kCODEC\_ModuleLinein = 5U,
        - kCODEC\_ModuleLineout = 6U,
        - kCODEC\_ModuleVref = 7U,
        - kCODEC\_ModuleMicbias = 8U,
        - kCODEC\_ModuleMic = 9U,
        - kCODEC\_ModuleI2SIn = 10U,
        - kCODEC\_ModuleI2SOut = 11U,
        - kCODEC\_ModuleMixer = 12U }
      - audio codec module*
        - enum codec\_module\_ctrl\_cmd\_t { kCODEC\_ModuleSwitchI2SInInterface = 0U }
        - audio codec module control cmd*
          - enum {
            - kCODEC\_ModuleI2SInInterfacePCM = 0U,
            - kCODEC\_ModuleI2SInInterfaceDSD = 1U }
          - audio codec module digital interface*
            - enum {

```

kCODEC_RecordSourceDifferentialLine = 1U,
kCODEC_RecordSourceLineInput = 2U,
kCODEC_RecordSourceDifferentialMic = 4U,
kCODEC_RecordSourceDigitalMic = 8U,
kCODEC_RecordSourceSingleEndMic = 16U }

```

*audio codec module record source value*

- enum {

```

kCODEC_RecordChannelLeft1 = 1U,
kCODEC_RecordChannelLeft2 = 2U,
kCODEC_RecordChannelLeft3 = 4U,
kCODEC_RecordChannelRight1 = 1U,
kCODEC_RecordChannelRight2 = 2U,
kCODEC_RecordChannelRight3 = 4U,
kCODEC_RecordChannelDifferentialPositive1 = 1U,
kCODEC_RecordChannelDifferentialPositive2 = 2U,
kCODEC_RecordChannelDifferentialPositive3 = 4U,
kCODEC_RecordChannelDifferentialNegative1 = 8U,
kCODEC_RecordChannelDifferentialNegative2 = 16U,
kCODEC_RecordChannelDifferentialNegative3 = 32U }

```

*audio codec record channel*

- enum {

```

kCODEC_PlaySourcePGA = 1U,
kCODEC_PlaySourceInput = 2U,
kCODEC_PlaySourceDAC = 4U,
kCODEC_PlaySourceMixerIn = 1U,
kCODEC_PlaySourceMixerInLeft = 2U,
kCODEC_PlaySourceMixerInRight = 4U,
kCODEC_PlaySourceAux = 8U }

```

*audio codec module play source value*

- enum {

```

kCODEC_PlayChannelHeadphoneLeft = 1U,
kCODEC_PlayChannelHeadphoneRight = 2U,
kCODEC_PlayChannelSpeakerLeft = 4U,
kCODEC_PlayChannelSpeakerRight = 8U,
kCODEC_PlayChannelLineOutLeft = 16U,
kCODEC_PlayChannelLineOutRight = 32U,
kCODEC_PlayChannelLeft0 = 1U,
kCODEC_PlayChannelRight0 = 2U,
kCODEC_PlayChannelLeft1 = 4U,
kCODEC_PlayChannelRight1 = 8U,
kCODEC_PlayChannelLeft2 = 16U,
kCODEC_PlayChannelRight2 = 32U,
kCODEC_PlayChannelLeft3 = 64U,
kCODEC_PlayChannelRight3 = 128U }

```

*codec play channel*

- enum {

```
kCODEC_VolumeHeadphoneLeft = 1U,
kCODEC_VolumeHeadphoneRight = 2U,
kCODEC_VolumeSpeakerLeft = 4U,
kCODEC_VolumeSpeakerRight = 8U,
kCODEC_VolumeLineOutLeft = 16U,
kCODEC_VolumeLineOutRight = 32U,
kCODEC_VolumeLeft0 = 1UL << 0U,
kCODEC_VolumeRight0 = 1UL << 1U,
kCODEC_VolumeLeft1 = 1UL << 2U,
kCODEC_VolumeRight1 = 1UL << 3U,
kCODEC_VolumeLeft2 = 1UL << 4U,
kCODEC_VolumeRight2 = 1UL << 5U,
kCODEC_VolumeLeft3 = 1UL << 6U,
kCODEC_VolumeRight3 = 1UL << 7U,
kCODEC_VolumeDAC = 1UL << 8U }
```

*codec volume setting*

- enum {

```

kCODEC_SupportModuleADC = 1U << 0U,
kCODEC_SupportModuleDAC = 1U << 1U,
kCODEC_SupportModulePGA = 1U << 2U,
kCODEC_SupportModuleHeadphone = 1U << 3U,
kCODEC_SupportModuleSpeaker = 1U << 4U,
kCODEC_SupportModuleLinein = 1U << 5U,
kCODEC_SupportModuleLineout = 1U << 6U,
kCODEC_SupportModuleVref = 1U << 7U,
kCODEC_SupportModuleMicbias = 1U << 8U,
kCODEC_SupportModuleMic = 1U << 9U,
kCODEC_SupportModuleI2SIn = 1U << 10U,
kCODEC_SupportModuleI2SOut = 1U << 11U,
kCODEC_SupportModuleMixer = 1U << 12U,
kCODEC_SupportModuleI2SInSwitchInterface = 1U << 13U,
kCODEC_SupportPlayChannelLeft0 = 1U << 0U,
kCODEC_SupportPlayChannelRight0 = 1U << 1U,
kCODEC_SupportPlayChannelLeft1 = 1U << 2U,
kCODEC_SupportPlayChannelRight1 = 1U << 3U,
kCODEC_SupportPlayChannelLeft2 = 1U << 4U,
kCODEC_SupportPlayChannelRight2 = 1U << 5U,
kCODEC_SupportPlayChannelLeft3 = 1U << 6U,
kCODEC_SupportPlayChannelRight3 = 1U << 7U,
kCODEC_SupportPlaySourcePGA = 1U << 8U,
kCODEC_SupportPlaySourceInput = 1U << 9U,
kCODEC_SupportPlaySourceDAC = 1U << 10U,
kCODEC_SupportPlaySourceMixerIn = 1U << 11U,
kCODEC_SupportPlaySourceMixerInLeft = 1U << 12U,
kCODEC_SupportPlaySourceMixerInRight = 1U << 13U,
kCODEC_SupportPlaySourceAux = 1U << 14U,
kCODEC_SupportRecordSourceDifferentialLine = 1U << 0U,
kCODEC_SupportRecordSourceLineInput = 1U << 1U,
kCODEC_SupportRecordSourceDifferentialMic = 1U << 2U,
kCODEC_SupportRecordSourceDigitalMic = 1U << 3U,
kCODEC_SupportRecordSourceSingleEndMic = 1U << 4U,
kCODEC_SupportRecordChannelLeft1 = 1U << 6U,
kCODEC_SupportRecordChannelLeft2 = 1U << 7U,
kCODEC_SupportRecordChannelLeft3 = 1U << 8U,
kCODEC_SupportRecordChannelRight1 = 1U << 9U,
kCODEC_SupportRecordChannelRight2 = 1U << 10U,
kCODEC_SupportRecordChannelRight3 = 1U << 11U }

```

*audio codec capability*

## Functions

- `status_t CODEC_Init` (`codec_handle_t *handle`, `codec_config_t *config`)  
*Codec initialization.*
- `status_t CODEC_Deinit` (`codec_handle_t *handle`)  
*Codec de-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 recordSource`)  
*codec set record source.*
- `status_t CODEC_SetRecordChannel` (`codec_handle_t *handle`, `uint32_t leftRecordChannel`, `uint32_t rightRecordChannel`)  
*codec set record channel.*
- `status_t CODEC_SetPlay` (`codec_handle_t *handle`, `uint32_t playSource`)  
*codec set play source.*

## Driver version

- `#define FSL_CODEC_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))`  
*CLOCK driver version 2.3.1.*

## 54.2.2 Data Structure Documentation

### 54.2.2.1 struct codec\_config\_t

#### Data Fields

- `uint32_t codecDevType`  
*codec type*
- `void * codecDevConfig`  
*Codec device specific configuration.*

### 54.2.2.2 struct codec\_capability\_t

#### Data Fields

- uint32\_t [codecModuleCapability](#)  
*codec module capability*
- uint32\_t [codecPlayCapability](#)  
*codec play capability*
- uint32\_t [codecRecordCapability](#)  
*codec record capability*
- uint32\_t [codecVolumeCapability](#)  
*codec volume capability*

### 54.2.2.3 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*

## 54.2.3 Macro Definition Documentation

### 54.2.3.1 #define FSL\_CODEC\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 1))

## 54.2.4 Enumeration Type Documentation

### 54.2.4.1 anonymous enum

Enumerator

***kStatus\_CODEC\_NotSupport*** CODEC not support status.

***kStatus\_CODEC\_DeviceNotRegistered*** CODEC device register failed status.

***kStatus\_CODEC\_I2CBusInitialFailed*** CODEC i2c bus initialization failed status.

***kStatus\_CODEC\_I2CCommandTransferFailed*** CODEC i2c bus command transfer failed status.

#### 54.2.4.2 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.

#### 54.2.4.3 anonymous enum

Enumerator

*kCODEC\_AudioSampleRate8KHz* Sample rate 8000 Hz.  
*kCODEC\_AudioSampleRate11025Hz* Sample rate 11025 Hz.  
*kCODEC\_AudioSampleRate12KHz* Sample rate 12000 Hz.  
*kCODEC\_AudioSampleRate16KHz* Sample rate 16000 Hz.  
*kCODEC\_AudioSampleRate22050Hz* Sample rate 22050 Hz.  
*kCODEC\_AudioSampleRate24KHz* Sample rate 24000 Hz.  
*kCODEC\_AudioSampleRate32KHz* Sample rate 32000 Hz.  
*kCODEC\_AudioSampleRate44100Hz* Sample rate 44100 Hz.  
*kCODEC\_AudioSampleRate48KHz* Sample rate 48000 Hz.  
*kCODEC\_AudioSampleRate96KHz* Sample rate 96000 Hz.  
*kCODEC\_AudioSampleRate192KHz* Sample rate 192000 Hz.  
*kCODEC\_AudioSampleRate384KHz* Sample rate 384000 Hz.

#### 54.2.4.4 anonymous enum

Enumerator

*kCODEC\_AudioBitWidth16bit* audio bit width 16  
*kCODEC\_AudioBitWidth20bit* audio bit width 20  
*kCODEC\_AudioBitWidth24bit* audio bit width 24  
*kCODEC\_AudioBitWidth32bit* audio bit width 32

#### 54.2.4.5 enum codec\_module\_t

Enumerator

*kCODEC\_ModuleADC* codec module ADC  
*kCODEC\_ModuleDAC* codec module DAC  
*kCODEC\_ModulePGA* codec module PGA  
*kCODEC\_ModuleHeadphone* codec module headphone



***kCODEC\_ModuleSpeaker*** codec module speaker  
***kCODEC\_ModuleLinein*** codec module linein  
***kCODEC\_ModuleLineout*** codec module lineout  
***kCODEC\_ModuleVref*** codec module VREF  
***kCODEC\_ModuleMicbias*** codec module MIC BIAS  
***kCODEC\_ModuleMic*** codec module MIC  
***kCODEC\_ModuleI2SIn*** codec module I2S in  
***kCODEC\_ModuleI2SOut*** codec module I2S out  
***kCODEC\_ModuleMixer*** codec module mixer

#### 54.2.4.6 enum codec\_module\_ctrl\_cmd\_t

Enumerator

***kCODEC\_ModuleSwitchI2SInInterface*** module digital interface swtch.

#### 54.2.4.7 anonymous enum

Enumerator

***kCODEC\_ModuleI2SInInterfacePCM*** Pcm interface.  
***kCODEC\_ModuleI2SInInterfaceDSD*** DSD interface.

#### 54.2.4.8 anonymous enum

Enumerator

***kCODEC\_RecordSourceDifferentialLine*** record source from differential line  
***kCODEC\_RecordSourceLineInput*** record source from line input  
***kCODEC\_RecordSourceDifferentialMic*** record source from differential mic  
***kCODEC\_RecordSourceDigitalMic*** record source from digital microphone  
***kCODEC\_RecordSourceSingleEndMic*** record source from single microphone

#### 54.2.4.9 anonymous enum

Enumerator

***kCODEC\_RecordChannelLeft1*** left record channel 1  
***kCODEC\_RecordChannelLeft2*** left record channel 2  
***kCODEC\_RecordChannelLeft3*** left record channel 3  
***kCODEC\_RecordChannelRight1*** right record channel 1  
***kCODEC\_RecordChannelRight2*** right record channel 2  
***kCODEC\_RecordChannelRight3*** right record channel 3  
***kCODEC\_RecordChannelDifferentialPositive1*** differential positive record channel 1

|                                                  |                                        |
|--------------------------------------------------|----------------------------------------|
| <i>kCODEC_RecordChannelDifferentialPositive2</i> | differential positive record channel 2 |
| <i>kCODEC_RecordChannelDifferentialPositive3</i> | differential positive record channel 3 |
| <i>kCODEC_RecordChannelDifferentialNegative1</i> | differential negative record channel 1 |
| <i>kCODEC_RecordChannelDifferentialNegative2</i> | differential negative record channel 2 |
| <i>kCODEC_RecordChannelDifferentialNegative3</i> | differential negative record channel 3 |

#### 54.2.4.10 anonymous enum

Enumerator

|                                      |                             |
|--------------------------------------|-----------------------------|
| <i>kCODEC_PlaySourcePGA</i>          | play source PGA, bypass ADC |
| <i>kCODEC_PlaySourceInput</i>        | play source Input3          |
| <i>kCODEC_PlaySourceDAC</i>          | play source DAC             |
| <i>kCODEC_PlaySourceMixerIn</i>      | play source mixer in        |
| <i>kCODEC_PlaySourceMixerInLeft</i>  | play source mixer in left   |
| <i>kCODEC_PlaySourceMixerInRight</i> | play source mixer in right  |
| <i>kCODEC_PlaySourceAux</i>          | play source mixer in AUx    |

#### 54.2.4.11 anonymous enum

Enumerator

|                                         |                              |
|-----------------------------------------|------------------------------|
| <i>kCODEC_PlayChannelHeadphoneLeft</i>  | play channel headphone left  |
| <i>kCODEC_PlayChannelHeadphoneRight</i> | play channel headphone right |
| <i>kCODEC_PlayChannelSpeakerLeft</i>    | play channel speaker left    |
| <i>kCODEC_PlayChannelSpeakerRight</i>   | play channel speaker right   |
| <i>kCODEC_PlayChannelLineOutLeft</i>    | play channel lineout left    |
| <i>kCODEC_PlayChannelLineOutRight</i>   | play channel lineout right   |
| <i>kCODEC_PlayChannelLeft0</i>          | play channel left0           |
| <i>kCODEC_PlayChannelRight0</i>         | play channel right0          |
| <i>kCODEC_PlayChannelLeft1</i>          | play channel left1           |
| <i>kCODEC_PlayChannelRight1</i>         | play channel right1          |
| <i>kCODEC_PlayChannelLeft2</i>          | play channel left2           |
| <i>kCODEC_PlayChannelRight2</i>         | play channel right2          |
| <i>kCODEC_PlayChannelLeft3</i>          | play channel left3           |
| <i>kCODEC_PlayChannelRight3</i>         | play channel right3          |

#### 54.2.4.12 anonymous enum

Enumerator

|                                    |                        |
|------------------------------------|------------------------|
| <i>kCODEC_VolumeHeadphoneLeft</i>  | headphone left volume  |
| <i>kCODEC_VolumeHeadphoneRight</i> | headphone right volume |
| <i>kCODEC_VolumeSpeakerLeft</i>    | speaker left volume    |
| <i>kCODEC_VolumeSpeakerRight</i>   | speaker right volume   |

*kCODEC\_VolumeLineOutLeft* lineout left volume  
*kCODEC\_VolumeLineOutRight* lineout right volume  
*kCODEC\_VolumeLeft0* left0 volume  
*kCODEC\_VolumeRight0* right0 volume  
*kCODEC\_VolumeLeft1* left1 volume  
*kCODEC\_VolumeRight1* right1 volume  
*kCODEC\_VolumeLeft2* left2 volume  
*kCODEC\_VolumeRight2* right2 volume  
*kCODEC\_VolumeLeft3* left3 volume  
*kCODEC\_VolumeRight3* right3 volume  
*kCODEC\_VolumeDAC* dac volume

#### 54.2.4.13 anonymous enum

Enumerator

*kCODEC\_SupportModuleADC* codec capability of module ADC  
*kCODEC\_SupportModuleDAC* codec capability of module DAC  
*kCODEC\_SupportModulePGA* codec capability of module PGA  
*kCODEC\_SupportModuleHeadphone* codec capability of module headphone  
*kCODEC\_SupportModuleSpeaker* codec capability of module speaker  
*kCODEC\_SupportModuleLinein* codec capability of module linein  
*kCODEC\_SupportModuleLineout* codec capability of module lineout  
*kCODEC\_SupportModuleVref* codec capability of module vref  
*kCODEC\_SupportModuleMicbias* codec capability of module mic bias  
*kCODEC\_SupportModuleMic* codec capability of module mic bias  
*kCODEC\_SupportModuleI2SIn* codec capability of module I2S in  
*kCODEC\_SupportModuleI2SOut* codec capability of module I2S out  
*kCODEC\_SupportModuleMixer* codec capability of module mixer  
*kCODEC\_SupportModuleI2SInSwitchInterface* codec capability of module I2S in switch interface

*kCODEC\_SupportPlayChannelLeft0* codec capability of play channel left 0  
*kCODEC\_SupportPlayChannelRight0* codec capability of play channel right 0  
*kCODEC\_SupportPlayChannelLeft1* codec capability of play channel left 1  
*kCODEC\_SupportPlayChannelRight1* codec capability of play channel right 1  
*kCODEC\_SupportPlayChannelLeft2* codec capability of play channel left 2  
*kCODEC\_SupportPlayChannelRight2* codec capability of play channel right 2  
*kCODEC\_SupportPlayChannelLeft3* codec capability of play channel left 3  
*kCODEC\_SupportPlayChannelRight3* codec capability of play channel right 3  
*kCODEC\_SupportPlaySourcePGA* codec capability of set playback source PGA  
*kCODEC\_SupportPlaySourceInput* codec capability of set playback source INPUT  
*kCODEC\_SupportPlaySourceDAC* codec capability of set playback source DAC  
*kCODEC\_SupportPlaySourceMixerIn* codec capability of set play source Mixer in  
*kCODEC\_SupportPlaySourceMixerInLeft* codec capability of set play source Mixer in left  
*kCODEC\_SupportPlaySourceMixerInRight* codec capability of set play source Mixer in right

*kCODEC\_SupportPlaySourceAux* codec capability of set play source aux

*kCODEC\_SupportRecordSourceDifferentialLine* codec capability of record source differential line

*kCODEC\_SupportRecordSourceLineInput* codec capability of record source line input

*kCODEC\_SupportRecordSourceDifferentialMic* codec capability of record source differential mic

*kCODEC\_SupportRecordSourceDigitalMic* codec capability of record digital mic

*kCODEC\_SupportRecordSourceSingleEndMic* codec capability of single end mic

*kCODEC\_SupportRecordChannelLeft1* left record channel 1

*kCODEC\_SupportRecordChannelLeft2* left record channel 2

*kCODEC\_SupportRecordChannelLeft3* left record channel 3

*kCODEC\_SupportRecordChannelRight1* right record channel 1

*kCODEC\_SupportRecordChannelRight2* right record channel 2

*kCODEC\_SupportRecordChannelRight3* right record channel 3

## 54.2.5 Function Documentation

### 54.2.5.1 `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.

### 54.2.5.2 `status_t CODEC_Deinit ( codec_handle_t * handle )`

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

### 54.2.5.3 `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.

#### 54.2.5.4 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.

#### 54.2.5.5 status\_t CODEC\_SetVolume ( codec\_handle\_t \* handle, uint32\_t channel, uint32\_t volume )

## Parameters

|                |                                                                                                                                            |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>  | codec handle.                                                                                                                              |
| <i>channel</i> | audio codec volume channel, can be a value or combine value of <code>_codec_volume_capability</code> or <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.

**54.2.5.6** `status_t CODEC_SetMute ( codec_handle_t * handle, uint32_t channel, bool mute )`

Parameters

|                |                                                                                                                                            |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>  | codec handle.                                                                                                                              |
| <i>channel</i> | audio codec volume channel, can be a value or combine value of <code>_codec_volume_capability</code> or <code>_codec_play_channel</code> . |
| <i>mute</i>    | true is mute, false is unmute.                                                                                                             |

Returns

`kStatus_Success` is success, else configure failed.

**54.2.5.7** `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.

**54.2.5.8** `status_t CODEC_SetRecord ( codec_handle_t * handle, uint32_t recordSource )`

## Parameters

|                     |                                                                                                   |
|---------------------|---------------------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                                     |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of <code>_codec_record_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

#### 54.2.5.9 `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 <code>_codec_record_channel</code> , can be a value combine of member in <code>_codec_record_channel</code> . |
| <i>rightRecord-Channel</i> | audio codec record channel, reference <code>_codec_record_channel</code> , can be a value combine of member in <code>_codec_record_channel</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

#### 54.2.5.10 `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 <code>_codec_play_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

## 54.3 CODEC I2C Driver

### 54.3.1 Overview

The codec common driver provides a codec control abstraction interface.

#### Data Structures

- struct `codec_i2c_config_t`  
*CODEC I2C configurations structure. [More...](#)*

#### Macros

- #define `CODEC_I2C_MASTER_HANDLER_SIZE` `HAL_I2C_MASTER_HANDLE_SIZE`  
*codec i2c handler*

#### Enumerations

- enum `codec_reg_addr_t` {  
`kCODEC_RegAddr8Bit` = 1U,  
`kCODEC_RegAddr16Bit` = 2U }  
*CODEC device register address type.*
- enum `codec_reg_width_t` {  
`kCODEC_RegWidth8Bit` = 1U,  
`kCODEC_RegWidth16Bit` = 2U,  
`kCODEC_RegWidth32Bit` = 4U }  
*CODEC device register width.*

#### Functions

- `status_t CODEC_I2C_Init` (void \*handle, uint32\_t i2cInstance, uint32\_t i2cBaudrate, uint32\_t i2cSourceClockHz)  
*Codec i2c bus initialization.*
- `status_t CODEC_I2C_Deinit` (void \*handle)  
*Codec i2c de-initialization.*
- `status_t CODEC_I2C_Send` (void \*handle, uint8\_t deviceAddress, uint32\_t subAddress, uint8\_t subaddressSize, uint8\_t \*txBuff, uint8\_t txBuffSize)  
*codec i2c send function.*
- `status_t CODEC_I2C_Receive` (void \*handle, uint8\_t deviceAddress, uint32\_t subAddress, uint8\_t subaddressSize, uint8\_t \*rxBuff, uint8\_t rxBuffSize)  
*codec i2c receive function.*



## 54.3.2 Data Structure Documentation

### 54.3.2.1 struct codec\_i2c\_config\_t

#### Data Fields

- uint32\_t [codecI2CInstance](#)  
*i2c bus instance*
- uint32\_t [codecI2CSourceClock](#)  
*i2c bus source clock frequency*

## 54.3.3 Enumeration Type Documentation

### 54.3.3.1 enum codec\_reg\_addr\_t

Enumerator

- kCODEC\_RegAddr8Bit* 8-bit register address.  
*kCODEC\_RegAddr16Bit* 16-bit register address.

### 54.3.3.2 enum codec\_reg\_width\_t

Enumerator

- kCODEC\_RegWidth8Bit* 8-bit register width.  
*kCODEC\_RegWidth16Bit* 16-bit register width.  
*kCODEC\_RegWidth32Bit* 32-bit register width.

## 54.3.4 Function Documentation

### 54.3.4.1 status\_t CODEC\_I2C\_Init ( void \* *handle*, uint32\_t *i2cInstance*, uint32\_t *i2cBaudrate*, uint32\_t *i2cSourceClockHz* )

Parameters

|                    |                                                                     |
|--------------------|---------------------------------------------------------------------|
| <i>handle</i>      | i2c master handle.                                                  |
| <i>i2cInstance</i> | instance number of the i2c bus, such as 0 is corresponding to I2C0. |

|                          |                             |
|--------------------------|-----------------------------|
| <i>i2cBaudrate</i>       | i2c baudrate.               |
| <i>i2cSource-ClockHz</i> | i2c source clock frequency. |

Returns

kStatus\_HAL\_I2cSuccess is success, else initial failed.

#### 54.3.4.2 status\_t CODEC\_I2C\_Deinit ( void \* *handle* )

Parameters

|               |                    |
|---------------|--------------------|
| <i>handle</i> | i2c master handle. |
|---------------|--------------------|

Returns

kStatus\_HAL\_I2cSuccess is success, else deinitial failed.

#### 54.3.4.3 status\_t CODEC\_I2C\_Send ( void \* *handle*, uint8\_t *deviceAddress*, uint32\_t *subAddress*, uint8\_t *subaddressSize*, uint8\_t \* *txBuff*, uint8\_t *txBuffSize* )

Parameters

|                       |                         |
|-----------------------|-------------------------|
| <i>handle</i>         | i2c master handle.      |
| <i>deviceAddress</i>  | codec device address.   |
| <i>subAddress</i>     | register address.       |
| <i>subaddressSize</i> | register address width. |
| <i>txBuff</i>         | tx buffer pointer.      |
| <i>txBuffSize</i>     | tx buffer size.         |

Returns

kStatus\_HAL\_I2cSuccess is success, else send failed.

#### 54.3.4.4 status\_t CODEC\_I2C\_Receive ( void \* *handle*, uint8\_t *deviceAddress*, uint32\_t *subAddress*, uint8\_t *subaddressSize*, uint8\_t \* *rxBuff*, uint8\_t *rxBuffSize* )

## Parameters

|                       |                         |
|-----------------------|-------------------------|
| <i>handle</i>         | i2c master handle.      |
| <i>deviceAddress</i>  | codec device address.   |
| <i>subAddress</i>     | register address.       |
| <i>subaddressSize</i> | register address width. |
| <i>rxBuff</i>         | rx buffer pointer.      |
| <i>rxBuffSize</i>     | rx buffer size.         |

## Returns

kStatus\_HAL\_I2cSuccess is success, else receive failed.

## 54.4 CS42888 Driver

### 54.4.1 Overview

The cs42888 driver provides a codec control interface.

### Data Structures

- struct `cs42888_audio_format_t`  
*cs42888 audio format [More...](#)*
- struct `cs42888_config_t`  
*Initialize structure of CS42888. [More...](#)*
- struct `cs42888_handle_t`  
*cs42888 handler [More...](#)*

### Macros

- #define `CS42888_I2C_HANDLER_SIZE` `CODEC_I2C_MASTER_HANDLER_SIZE`  
*CS42888 handle size.*
- #define `CS42888_ID` `0x01U`  
*Define the register address of CS42888.*
- #define `CS42888_AOUT_MAX_VOLUME_VALUE` `0xFFU`  
*CS42888 volume setting range.*
- #define `CS42888_CACHEREGNUM` `28U`  
*Cache register number.*
- #define `CS42888_I2C_ADDR` `0x48U`  
*CS42888 I2C address.*
- #define `CS42888_I2C_BITRATE` `(100000U)`  
*CS42888 I2C baudrate.*

### Typedefs

- typedef void(\* `cs42888_reset`)(bool state)  
*cs42888 reset function pointer*

### Enumerations

- enum `cs42888_func_mode` {  
`kCS42888_ModeMasterSSM = 0x0,`  
`kCS42888_ModeMasterDSM = 0x1,`  
`kCS42888_ModeMasterQSM = 0x2,`  
`kCS42888_ModeSlave = 0x3 }`  
*CS42888 support modes.*

- enum `cs42888_module_t` {  
`kCS42888_ModuleDACPair1 = 0x2,`  
`kCS42888_ModuleDACPair2 = 0x4,`  
`kCS42888_ModuleDACPair3 = 0x8,`  
`kCS42888_ModuleDACPair4 = 0x10,`  
`kCS42888_ModuleADCPair1 = 0x20,`  
`kCS42888_ModuleADCPair2 = 0x40 }`  
*Modules in CS42888 board.*
- enum `cs42888_bus_t` {  
`kCS42888_BusLeftJustified = 0x0,`  
`kCS42888_BusI2S = 0x1,`  
`kCS42888_BusRightJustified = 0x2,`  
`kCS42888_BusOL1 = 0x4,`  
`kCS42888_BusOL2 = 0x5,`  
`kCS42888_BusTDM = 0x6 }`  
*CS42888 supported audio bus type.*
- enum {  
`kCS42888_AOUT1 = 1U,`  
`kCS42888_AOUT2 = 2U,`  
`kCS42888_AOUT3 = 3U,`  
`kCS42888_AOUT4 = 4U,`  
`kCS42888_AOUT5 = 5U,`  
`kCS42888_AOUT6 = 6U,`  
`kCS42888_AOUT7 = 7U,`  
`kCS42888_AOUT8 = 8U }`  
*CS428888 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 sampleRate, uint32_t bits)`
- *Configure the data format of audio data.*
- `status_t CS42888_WriteReg (cs42888_handle_t *handle, uint8_t reg, uint8_t val)`
- *Write register to CS42888 using I2C.*
- `status_t CS42888_ReadReg (cs42888_handle_t *handle, uint8_t reg, uint8_t *val)`
- *Read register from CS42888 using I2C.*
- `status_t CS42888_ModifyReg (cs42888_handle_t *handle, uint8_t reg, uint8_t mask, uint8_t val)`
- *Modify some bits in the register using I2C.*

## Driver version

- `#define FSL_CS42888_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))`  
*cs42888 driver version 2.1.3.*

## 54.4.2 Data Structure Documentation

### 54.4.2.1 struct cs42888\_audio\_format\_t

#### Data Fields

- `uint32_t mclk_HZ`  
*master clock frequency*
- `uint32_t sampleRate`  
*sample rate*
- `uint32_t bitWidth`  
*bit width*

### 54.4.2.2 struct cs42888\_config\_t

#### Data Fields

- `cs42888_bus_t bus`  
*Audio transfer protocol.*
- `cs42888_audio_format_t format`  
*cs42888 audio format*

- `cs42888_func_mode` ADCMode  
*CS42888 ADC function mode.*
- `cs42888_func_mode` DACMode  
*CS42888 DAC function mode.*
- `bool` `master`  
*true is master, false is slave*
- `codec_i2c_config_t` `i2cConfig`  
*i2c bus configuration*
- `uint8_t` `slaveAddress`  
*slave address*
- `cs42888_reset` `reset`  
*reset function pointer*

### Field Documentation

(1) `cs42888_func_mode` `cs42888_config_t::ADCMode`

(2) `cs42888_func_mode` `cs42888_config_t::DACMode`

#### 54.4.2.3 struct `cs42888_handle_t`

### Data Fields

- `cs42888_config_t * config`  
*cs42888 config pointer*
- `uint8_t i2cHandle` [`CS42888_I2C_HANDLER_SIZE`]  
*i2c handle pointer*

### 54.4.3 Macro Definition Documentation

54.4.3.1 `#define FSL_CS42888_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))`

54.4.3.2 `#define CS42888_ID 0x01U`

54.4.3.3 `#define CS42888_I2C_ADDR 0x48U`

### 54.4.4 Enumeration Type Documentation

#### 54.4.4.1 enum `cs42888_func_mode`

Enumerator

*kCS42888\_ModeMasterSSM* master single speed mode  
*kCS42888\_ModeMasterDSM* master dual speed mode  
*kCS42888\_ModeMasterQSM* master quad speed mode  
*kCS42888\_ModeSlave* master single speed mode

#### 54.4.4.2 enum cs42888\_module\_t

Enumerator

*kCS42888\_ModuleDACPair1* DAC pair1 (AOUT1 and AOUT2) module in CS42888.  
*kCS42888\_ModuleDACPair2* DAC pair2 (AOUT3 and AOUT4) module in CS42888.  
*kCS42888\_ModuleDACPair3* DAC pair3 (AOUT5 and AOUT6) module in CS42888.  
*kCS42888\_ModuleDACPair4* DAC pair4 (AOUT7 and AOUT8) module in CS42888.  
*kCS42888\_ModuleADCPair1* ADC pair1 (AIN1 and AIN2) module in CS42888.  
*kCS42888\_ModuleADCPair2* ADC pair2 (AIN3 and AIN4) module in CS42888.

#### 54.4.4.3 enum cs42888\_bus\_t

Enumerator

*kCS42888\_BusLeftJustified* Left justified format, up to 24 bits.  
*kCS42888\_BusI2S* I2S format, up to 24 bits.  
*kCS42888\_BusRightJustified* Right justified, can support 16bits and 24 bits.  
*kCS42888\_BusOL1* One-Line #1 mode.  
*kCS42888\_BusOL2* One-Line #2 mode.  
*kCS42888\_BusTDM* TDM mode.

#### 54.4.4.4 anonymous enum

Enumerator

*kCS42888\_AOUT1* aout1  
*kCS42888\_AOUT2* aout2  
*kCS42888\_AOUT3* aout3  
*kCS42888\_AOUT4* aout4  
*kCS42888\_AOUT5* aout5  
*kCS42888\_AOUT6* aout6  
*kCS42888\_AOUT7* aout7  
*kCS42888\_AOUT8* aout8

### 54.4.5 Function Documentation

#### 54.4.5.1 status\_t CS42888\_Init ( cs42888\_handle\_t \* handle, cs42888\_config\_t \* config )

The second parameter is NULL to CS42888 in this version. If users want to change the settings, they have to use `cs42888_write_reg()` or `cs42888_modify_reg()` to set the register value of CS42888. Note: If the `codec_config` is NULL, it would initialize CS42888 using default settings. The default setting: `codec_config->bus = kCS42888_BusI2S` `codec_config->ADCmode = kCS42888_ModeSlave` `codec_config->DACmode = kCS42888_ModeSlave`



Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>handle</i> | CS42888 handle structure.        |
| <i>config</i> | CS42888 configuration structure. |

#### 54.4.5.2 `status_t CS42888_Deinit ( cs42888_handle_t * handle )`

This function close all modules in CS42888 to save power.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>handle</i> | CS42888 handle structure pointer. |
|---------------|-----------------------------------|

#### 54.4.5.3 `status_t CS42888_SetProtocol ( cs42888_handle_t * handle, cs42888_bus_t protocol, uint32_t bitWidth )`

CS42888 only supports I2S, left justified, right justified, PCM A, PCM B format.

Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>handle</i>   | CS42888 handle structure.     |
| <i>protocol</i> | Audio data transfer protocol. |
| <i>bitWidth</i> | bit width                     |

#### 54.4.5.4 `void CS42888_SetFuncMode ( cs42888_handle_t * handle, cs42888_func_mode mode )`

**Deprecated** api, Do not use it anymore. It has been superceded by [CS42888\\_SelectFunctionalMode](#).

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>handle</i> | CS42888 handle structure.           |
| <i>mode</i>   | differenht working mode of CS42888. |

#### 54.4.5.5 `status_t CS42888_SelectFunctionalMode ( cs42888_handle_t * handle, cs42888_func_mode adcMode, cs42888_func_mode dacMode )`

## Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>handle</i>  | CS42888 handle structure.          |
| <i>adcMode</i> | different working mode of CS42888. |
| <i>dacMode</i> | different working mode of CS42888. |

#### 54.4.5.6 **status\_t CS42888\_SetAOUTVolume ( cs42888\_handle\_t \* *handle*, uint8\_t *channel*, uint8\_t *volume* )**

This function would set the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

## Parameters

|                |                                |
|----------------|--------------------------------|
| <i>handle</i>  | CS42888 handle structure.      |
| <i>channel</i> | AOUT channel, it shall be 1~8. |
| <i>volume</i>  | Volume value need to be set.   |

#### 54.4.5.7 **status\_t CS42888\_SetAINVolume ( cs42888\_handle\_t \* *handle*, uint8\_t *channel*, uint8\_t *volume* )**

This function would set the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

## Parameters

|                |                               |
|----------------|-------------------------------|
| <i>handle</i>  | CS42888 handle structure.     |
| <i>channel</i> | AIN channel, it shall be 1~4. |
| <i>volume</i>  | Volume value need to be set.  |

#### 54.4.5.8 **uint8\_t CS42888\_GetAOUTVolume ( cs42888\_handle\_t \* *handle*, uint8\_t *channel* )**

This function gets the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

## Parameters

|                |                                |
|----------------|--------------------------------|
| <i>handle</i>  | CS42888 handle structure.      |
| <i>channel</i> | AOUT channel, it shall be 1~8. |

**54.4.5.9 uint8\_t CS42888\_GetAINVolume ( cs42888\_handle\_t \* *handle*, uint8\_t *channel* )**

This function gets the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

## Parameters

|                |                               |
|----------------|-------------------------------|
| <i>handle</i>  | CS42888 handle structure.     |
| <i>channel</i> | AIN channel, it shall be 1~4. |

**54.4.5.10 status\_t CS42888\_SetMute ( cs42888\_handle\_t \* *handle*, uint8\_t *channelMask* )**

## Parameters

|                    |                                                                                                                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>      | CS42888 handle structure.                                                                                                                                                                              |
| <i>channelMask</i> | Channel mask for mute. Mute channel 0, it shall be 0x1, while mute channel 0 and 1, it shall be 0x3. Mute all channel, it shall be 0xFF. Each bit represent one channel, 1 means mute, 0 means unmute. |

**54.4.5.11 status\_t CS42888\_SetChannelMute ( cs42888\_handle\_t \* *handle*, uint8\_t *channel*, bool *isMute* )**

## Parameters

|                |                                                |
|----------------|------------------------------------------------|
| <i>handle</i>  | CS42888 handle structure.                      |
| <i>channel</i> | reference <code>_cs42888_play_channel</code> . |
| <i>isMute</i>  | true is mute, false is unmute.                 |

**54.4.5.12 status\_t CS42888\_SetModule ( cs42888\_handle\_t \* *handle*, cs42888\_module\_t *module*, bool *isEnabled* )**

## Parameters

|                  |                            |
|------------------|----------------------------|
| <i>handle</i>    | CS42888 handle structure.  |
| <i>module</i>    | Module expected to enable. |
| <i>isEnabled</i> | Enable or disable moudles. |

#### 54.4.5.13 **status\_t CS42888\_ConfigDataFormat ( cs42888\_handle\_t \* *handle*, uint32\_t *mclk*, uint32\_t *sample\_rate*, uint32\_t *bits* )**

This function would configure the registers about the sample rate, bit depths.

## Parameters

|                    |                                                                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>      | CS42888 handle structure pointer.                                                                                                           |
| <i>mclk</i>        | Master clock frequency of I2S.                                                                                                              |
| <i>sample_rate</i> | Sample rate of audio file running in CS42888. CS42888 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate. |
| <i>bits</i>        | Bit depth of audio file (CS42888 only supports 16bit, 20bit, 24bit and 32 bit in HW).                                                       |

#### 54.4.5.14 **status\_t CS42888\_WriteReg ( cs42888\_handle\_t \* *handle*, uint8\_t *reg*, uint8\_t *val* )**

## Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>handle</i> | CS42888 handle structure.               |
| <i>reg</i>    | The register address in CS42888.        |
| <i>val</i>    | Value needs to write into the register. |

#### 54.4.5.15 **status\_t CS42888\_ReadReg ( cs42888\_handle\_t \* *handle*, uint8\_t *reg*, uint8\_t \* *val* )**

## Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>handle</i> | CS42888 handle structure.        |
| <i>reg</i>    | The register address in CS42888. |
| <i>val</i>    | Value written to.                |

**54.4.5.16** `status_t CS42888_ModifyReg ( cs42888_handle_t * handle, uint8_t reg, uint8_t mask, uint8_t val )`

Parameters

|               |                                                                                  |
|---------------|----------------------------------------------------------------------------------|
| <i>handle</i> | CS42888 handle structure.                                                        |
| <i>reg</i>    | The register address in CS42888.                                                 |
| <i>mask</i>   | The mask code for the bits want to write. The bit you want to write should be 0. |
| <i>val</i>    | Value needs to write into the register.                                          |

## 54.4.6 CS42888 Adapter

### 54.4.6.1 Overview

The cs42888 adapter provides a codec unify control interface.

#### Macros

- #define `HAL_CODEC_CS42888_HANDLER_SIZE` (`CS42888_I2C_HANDLER_SIZE` + 4)  
*codec handler size*

#### Functions

- `status_t HAL_CODEC_CS42888_Init` (void \*handle, void \*config)  
*Codec initialization.*
- `status_t HAL_CODEC_CS42888_Deinit` (void \*handle)  
*Codec de-initialization.*
- `status_t HAL_CODEC_CS42888_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- `status_t HAL_CODEC_CS42888_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- `status_t HAL_CODEC_CS42888_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- `status_t HAL_CODEC_CS42888_SetPower` (void \*handle, uint32\_t module, bool powerOn)  
*set audio codec module power.*
- `status_t HAL_CODEC_CS42888_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- `status_t HAL_CODEC_CS42888_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- `status_t HAL_CODEC_CS42888_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- `status_t HAL_CODEC_CS42888_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*
- static `status_t HAL_CODEC_Init` (void \*handle, void \*config)  
*Codec initialization.*
- static `status_t HAL_CODEC_Deinit` (void \*handle)  
*Codec de-initialization.*
- static `status_t HAL_CODEC_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- static `status_t HAL_CODEC_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- static `status_t HAL_CODEC_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- static `status_t HAL_CODEC_SetPower` (void \*handle, uint32\_t module, bool powerOn)

- *set audio codec module power.*  
static [status\\_t HAL\\_CODEC\\_SetRecord](#) (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- static [status\\_t HAL\\_CODEC\\_SetRecordChannel](#) (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- static [status\\_t HAL\\_CODEC\\_SetPlay](#) (void \*handle, uint32\_t playSource)  
*codec set play source.*
- static [status\\_t HAL\\_CODEC\\_ModuleControl](#) (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

## 54.4.6.2 Function Documentation

### 54.4.6.2.1 status\_t HAL\_CODEC\_CS42888\_Init ( void \* handle, void \* config )

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

Returns

kStatus\_Success is success, else initial failed.

### 54.4.6.2.2 status\_t HAL\_CODEC\_CS42888\_Deinit ( void \* handle )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

### 54.4.6.2.3 status\_t HAL\_CODEC\_CS42888\_SetFormat ( void \* handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth )

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.4.6.2.4 status\_t HAL\_CODEC\_CS42888\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

## Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.4.6.2.5 status\_t HAL\_CODEC\_CS42888\_SetMute ( void \* *handle*, uint32\_t *playChannel*, bool *isMute* )

## Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.4.6.2.6 status\_t HAL\_CODEC\_CS42888\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* )



## Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.4.6.2.7 status\_t HAL\_CODEC\_CS42888\_SetRecord ( void \* *handle*, uint32\_t *recordSource* )

## Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.4.6.2.8 status\_t HAL\_CODEC\_CS42888\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* )

## Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.4.6.2.9 status\_t HAL\_CODEC\_CS42888\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

## Parameters

|                   |                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                                 |
| <i>playSource</i> | audio codec play source, can be a value or combine value of <code>_codec_play_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

#### 54.4.6.2.10 `status_t HAL_CODEC_CS42888_ModuleControl ( void * handle, uint32_t cmd, uint32_t data )`

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

## Parameters

|               |                                                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                                                     |
| <i>cmd</i>    | module control cmd, reference <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.

#### 54.4.6.2.11 `static status_t HAL_CODEC_Init ( void * handle, void * config ) [inline], [static]`

## Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

## Returns

`kStatus_Success` is success, else initial failed.

#### 54.4.6.2.12 `static status_t HAL_CODEC_Deinit ( void * handle ) [inline], [static]`

## Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

## Returns

kStatus\_Success is success, else de-initial failed.

**54.4.6.2.13** `static status_t HAL_CODEC_SetFormat ( void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth ) [inline], [static]`

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

## Returns

kStatus\_Success is success, else configure failed.

**54.4.6.2.14** `static status_t HAL_CODEC_SetVolume ( void * handle, uint32_t playChannel, uint32_t volume ) [inline], [static]`

## Parameters

|                    |                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                                   |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of <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.

**54.4.6.2.15** `static status_t HAL_CODEC_SetMute ( void * handle, uint32_t playChannel, bool isMute ) [inline], [static]`

## Parameters

|                    |                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                                   |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> . |
| <i>isMute</i>      | true is mute, false is unmute.                                                                  |

## Returns

`kStatus_Success` is success, else configure failed.

**54.4.6.2.16** `static status_t HAL_CODEC_SetPower ( void * handle, uint32_t module, bool powerOn ) [inline], [static]`

## Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

## Returns

`kStatus_Success` is success, else configure failed.

**54.4.6.2.17** `static status_t HAL_CODEC_SetRecord ( void * handle, uint32_t recordSource ) [inline], [static]`

## Parameters

|                     |                                                                                                   |
|---------------------|---------------------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                                     |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of <code>_codec_record_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

**54.4.6.2.18** `static status_t HAL_CODEC_SetRecordChannel ( void * handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel ) [inline], [static]`

## Parameters

|                            |                                                                                                                                                              |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                                                |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference <code>_codec_record_channel</code> , can be a value or combine value of member in <code>_codec_record_channel</code> . |
| <i>rightRecord-Channel</i> | audio codec record channel, reference <code>_codec_record_channel</code> , can be a value or combine value of member in <code>_codec_record_channel</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

**54.4.6.2.19** `static status_t HAL_CODEC_SetPlay ( void * handle, uint32_t playSource ) [inline], [static]`

## Parameters

|                   |                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                                 |
| <i>playSource</i> | audio codec play source, can be a value or combine value of <code>_codec_play_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

**54.4.6.2.20** `static status_t HAL_CODEC_ModuleControl ( void * handle, uint32_t cmd, uint32_t data ) [inline], [static]`

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

## Parameters

|               |                                                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                                                     |
| <i>cmd</i>    | module control cmd, reference <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.

## 54.5 DA7212 Driver

### 54.5.1 Overview

The da7212 driver provides a codec control interface.

### Data Structures

- struct `da7212_pll_config_t`  
*da7212 pll configuration [More...](#)*
- struct `da7212_audio_format_t`  
*da7212 audio format [More...](#)*
- struct `da7212_config_t`  
*DA7212 configure structure. [More...](#)*
- struct `da7212_handle_t`  
*da7212 codec handler [More...](#)*

### Macros

- #define `DA7212_I2C_HANDLER_SIZE` `CODEC_I2C_MASTER_HANDLER_SIZE`  
*da7212 handle size*
- #define `DA7212_ADDRESS` `(0x1A)`  
*DA7212 I2C address.*
- #define `DA7212_HEADPHONE_MAX_VOLUME_VALUE` `0x3FU`  
*da7212 volume setting range*

### Enumerations

- enum `da7212_input_t` {  
  `kDA7212_Input_AUX` = 0x0,  
  `kDA7212_Input_MIC1_Dig`,  
  `kDA7212_Input_MIC1_An`,  
  `kDA7212_Input_MIC2` }  
*DA7212 input source select.*
- enum `_da7212_play_channel` {  
  `kDA7212_HeadphoneLeft` = 1U,  
  `kDA7212_HeadphoneRight` = 2U,  
  `kDA7212_Speaker` = 4U }  
*da7212 play channel*
- enum `da7212_output_t` {  
  `kDA7212_Output_HP` = 0x0,  
  `kDA7212_Output_SP` }  
*DA7212 output device select.*

- enum `_da7212_module` {  
`kDA7212_ModuleADC`,  
`kDA7212_ModuleDAC`,  
`kDA7212_ModuleHeadphone`,  
`kDA7212_ModuleSpeaker` }  
*DA7212 module.*
- enum `da7212_dac_source_t` {  
`kDA7212_DACSourceADC` = 0x0U,  
`kDA7212_DACSourceInputStream` = 0x3U }  
*DA7212 functionality.*
- enum `da7212_volume_t` {  
`kDA7212_DACGainMute` = 0x7,  
`kDA7212_DACGainM72DB` = 0x17,  
`kDA7212_DACGainM60DB` = 0x1F,  
`kDA7212_DACGainM54DB` = 0x27,  
`kDA7212_DACGainM48DB` = 0x2F,  
`kDA7212_DACGainM42DB` = 0x37,  
`kDA7212_DACGainM36DB` = 0x3F,  
`kDA7212_DACGainM30DB` = 0x47,  
`kDA7212_DACGainM24DB` = 0x4F,  
`kDA7212_DACGainM18DB` = 0x57,  
`kDA7212_DACGainM12DB` = 0x5F,  
`kDA7212_DACGainM6DB` = 0x67,  
`kDA7212_DACGain0DB` = 0x6F,  
`kDA7212_DACGain6DB` = 0x77,  
`kDA7212_DACGain12DB` = 0x7F }  
*DA7212 volume.*
- enum `da7212_protocol_t` {  
`kDA7212_BusI2S` = 0x0,  
`kDA7212_BusLeftJustified`,  
`kDA7212_BusRightJustified`,  
`kDA7212_BusDSPMode` }  
*The audio data transfer protocol choice.*
- enum `da7212_sys_clk_source_t` {  
`kDA7212_SysClkSourceMCLK` = 0U,  
`kDA7212_SysClkSourcePLL` = 1U << 14 }  
*da7212 system clock source*
- enum `da7212_pll_clk_source_t` { `kDA7212_PLLClkSourceMCLK` = 0U }  
*DA7212 pll clock source.*
- enum `da7212_pll_out_clk_t` {  
`kDA7212_PLLOutputClk11289600` = 11289600U,  
`kDA7212_PLLOutputClk12288000` = 12288000U }  
*DA7212 output clock frequency.*
- enum `da7212_master_bits_t` {

```

kDA7212_MasterBits32PerFrame = 0U,
kDA7212_MasterBits64PerFrame = 1U,
kDA7212_MasterBits128PerFrame = 2U,
kDA7212_MasterBits256PerFrame = 3U }
 master mode bits per frame

```

## Functions

- `status_t DA7212_Init (da7212_handle_t *handle, da7212_config_t *codecConfig)`  
*DA7212 initialize function.*
- `status_t DA7212_ConfigAudioFormat (da7212_handle_t *handle, uint32_t masterClock_Hz, uint32_t sampleRate_Hz, uint32_t dataBits)`  
*Set DA7212 audio format.*
- `status_t DA7212_SetPLLConfig (da7212_handle_t *handle, da7212_pll_config_t *config)`  
*DA7212 set PLL configuration This function will enable the GPIO1 FLL clock output function, so user can see the generated 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 channel, uint32_t volume)`  
*Set module volume.*
- `status_t DA7212_SetChannelMute (da7212_handle_t *handle, uint32_t channel, bool isMute)`  
*Set module mute.*
- `status_t DA7212_SetProtocol (da7212_handle_t *handle, da7212_protocol_t protocol)`  
*Set protocol for DA7212.*
- `status_t DA7212_SetMasterModeBits (da7212_handle_t *handle, uint32_t bitWidth)`  
*Set master mode bits per frame for DA7212.*
- `status_t DA7212_WriteRegister (da7212_handle_t *handle, uint8_t u8Register, uint8_t u8RegisterData)`  
*Write a register for DA7212.*
- `status_t DA7212_ReadRegister (da7212_handle_t *handle, uint8_t u8Register, uint8_t *pu8RegisterData)`  
*Get a register value of DA7212.*
- `status_t DA7212_Deinit (da7212_handle_t *handle)`  
*Deinit DA7212.*

## Driver version

- `#define FSL_DA7212_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))`  
*CLOCK driver version 2.2.3.*



## 54.5.2 Data Structure Documentation

### 54.5.2.1 struct da7212\_pll\_config\_t

#### Data Fields

- [da7212\\_pll\\_clk\\_source\\_t](#) source  
*pll reference clock source*
- [uint32\\_t](#) [refClock\\_HZ](#)  
*pll reference clock frequency*
- [da7212\\_pll\\_out\\_clk\\_t](#) [outputClock\\_HZ](#)  
*pll output clock frequency*

### 54.5.2.2 struct da7212\_audio\_format\_t

#### Data Fields

- [uint32\\_t](#) [mclk\\_HZ](#)  
*master clock frequency*
- [uint32\\_t](#) [sampleRate](#)  
*sample rate*
- [uint32\\_t](#) [bitWidth](#)  
*bit width*
- [bool](#) [isBclkInvert](#)  
*bit clock intervnet*

### 54.5.2.3 struct da7212\_config\_t

#### Data Fields

- [bool](#) [isMaster](#)  
*If DA7212 is master, true means master, false means slave.*
- [da7212\\_protocol\\_t](#) [protocol](#)  
*Audio bus format, can be I2S, LJ, RJ or DSP mode.*
- [da7212\\_dac\\_source\\_t](#) [dacSource](#)  
*DA7212 data source.*
- [da7212\\_audio\\_format\\_t](#) [format](#)  
*audio format*
- [uint8\\_t](#) [slaveAddress](#)  
*device address*
- [codec\\_i2c\\_config\\_t](#) [i2cConfig](#)  
*i2c configuration*
- [da7212\\_sys\\_clk\\_source\\_t](#) [sysClkSource](#)  
*system clock source*
- [da7212\\_pll\\_config\\_t](#) \* [pll](#)  
*pll configuration*

## Field Documentation

- (1) `bool da7212_config_t::isMaster`
- (2) `da7212_protocol_t da7212_config_t::protocol`
- (3) `da7212_dac_source_t da7212_config_t::dacSource`

### 54.5.2.4 struct da7212\_handle\_t

#### Data Fields

- `da7212_config_t * config`  
*da7212 config pointer*
- `uint8_t i2cHandle [DA7212_I2C_HANDLER_SIZE]`  
*i2c handle*

### 54.5.3 Macro Definition Documentation

54.5.3.1 `#define FSL_DA7212_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))`

### 54.5.4 Enumeration Type Documentation

#### 54.5.4.1 enum da7212\_Input\_t

Enumerator

*kDA7212\_Input\_AUX* Input from AUX.  
*kDA7212\_Input\_MIC1\_Dig* Input from MIC1 Digital.  
*kDA7212\_Input\_MIC1\_An* Input from Mic1 Analog.  
*kDA7212\_Input\_MIC2* Input from MIC2.

#### 54.5.4.2 enum \_da7212\_play\_channel

Enumerator

*kDA7212\_HeadphoneLeft* headphone left  
*kDA7212\_HeadphoneRight* headphone right  
*kDA7212\_Speaker* speaker channel

#### 54.5.4.3 enum da7212\_Output\_t

Enumerator

*kDA7212\_Output\_HP* Output to headphone.

*kDA7212\_Output\_SP* Output to speaker.

#### 54.5.4.4 enum *da7212\_module*

Enumerator

*kDA7212\_ModuleADC* module ADC  
*kDA7212\_ModuleDAC* module DAC  
*kDA7212\_ModuleHeadphone* module headphone  
*kDA7212\_ModuleSpeaker* module speaker

#### 54.5.4.5 enum *da7212\_dac\_source\_t*

Enumerator

*kDA7212\_DACSourceADC* DAC source from ADC.  
*kDA7212\_DACSourceInputStream* DAC source from.

#### 54.5.4.6 enum *da7212\_volume\_t*

Enumerator

*kDA7212\_DACGainMute* Mute DAC.  
*kDA7212\_DACGainM72DB* DAC volume -72db.  
*kDA7212\_DACGainM60DB* DAC volume -60db.  
*kDA7212\_DACGainM54DB* DAC volume -54db.  
*kDA7212\_DACGainM48DB* DAC volume -48db.  
*kDA7212\_DACGainM42DB* DAC volume -42db.  
*kDA7212\_DACGainM36DB* DAC volume -36db.  
*kDA7212\_DACGainM30DB* DAC volume -30db.  
*kDA7212\_DACGainM24DB* DAC volume -24db.  
*kDA7212\_DACGainM18DB* DAC volume -18db.  
*kDA7212\_DACGainM12DB* DAC volume -12db.  
*kDA7212\_DACGainM6DB* DAC volume -6db.  
*kDA7212\_DACGain0DB* DAC volume +0db.  
*kDA7212\_DACGain6DB* DAC volume +6db.  
*kDA7212\_DACGain12DB* DAC volume +12db.

#### 54.5.4.7 enum *da7212\_protocol\_t*

Enumerator

*kDA7212\_BusI2S* I2S Type.

*kDA7212\_BusLeftJustified* Left justified.  
*kDA7212\_BusRightJustified* Right Justified.  
*kDA7212\_BusDSPMode* DSP mode.

#### 54.5.4.8 enum da7212\_sys\_clk\_source\_t

Enumerator

*kDA7212\_SysClkSourceMCLK* da7212 system clock source from MCLK  
*kDA7212\_SysClkSourcePLL* da7212 system clock source from PLL

#### 54.5.4.9 enum da7212\_pll\_clk\_source\_t

Enumerator

*kDA7212\_PLLClkSourceMCLK* DA7212 PLL clock source from MCLK.

#### 54.5.4.10 enum da7212\_pll\_out\_clk\_t

Enumerator

*kDA7212\_PLLOutputClk11289600U* output 112896000U  
*kDA7212\_PLLOutputClk12288000U* output 12288000U

#### 54.5.4.11 enum da7212\_master\_bits\_t

Enumerator

*kDA7212\_MasterBits32PerFrame* master mode bits32 per frame  
*kDA7212\_MasterBits64PerFrame* master mode bits64 per frame  
*kDA7212\_MasterBits128PerFrame* master mode bits128 per frame  
*kDA7212\_MasterBits256PerFrame* master mode bits256 per frame

### 54.5.5 Function Documentation

54.5.5.1 **status\_t DA7212\_Init ( da7212\_handle\_t \* *handle*, da7212\_config\_t \* *codecConfig* )**

## Parameters

|                    |                                                                                                                                                                                                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>      | DA7212 handle pointer.                                                                                                                                                                                                                                                        |
| <i>codecConfig</i> | Codec configure structure. This parameter can be NULL, if NULL, set as default settings. The default setting:<br><br><pre>* sgtl_init_t codec_config * codec_config.route = kDA7212_RoutePlayback * codec_config.bus = kDA7212_BusI2S * codec_config.isMaster = false *</pre> |

#### 54.5.5.2 `status_t DA7212_ConfigAudioFormat ( da7212_handle_t * handle, uint32_t masterClock_Hz, uint32_t sampleRate_Hz, uint32_t dataBits )`

## Parameters

|                       |                                                                                                                                                                                                                                       |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>         | DA7212 handle pointer.                                                                                                                                                                                                                |
| <i>masterClock_Hz</i> | Master clock frequency in Hz. If DA7212 is slave, use the frequency of master, if DA7212 as master, it should be 1228000 while sample rate frequency is 8k/12K/16K/24K/32K/48K/96K, 11289600 whie sample rate is 11.025K/22.05K/44.1K |
| <i>sampleRate_Hz</i>  | Sample rate frequency in Hz.                                                                                                                                                                                                          |
| <i>dataBits</i>       | How many bits in a word of a audio frame, DA7212 only supports 16/20/24/32 bits.                                                                                                                                                      |

#### 54.5.5.3 `status_t DA7212_SetPLLConfig ( da7212_handle_t * handle, da7212_pll_config_t * config )`

## Parameters

|               |                            |
|---------------|----------------------------|
| <i>handle</i> | DA7212 handler pointer.    |
| <i>config</i> | PLL configuration pointer. |

#### 54.5.5.4 `void DA7212_ChangeHPVolume ( da7212_handle_t * handle, da7212_volume_t volume )`

Parameters

|               |                         |
|---------------|-------------------------|
| <i>handle</i> | DA7212 handle pointer.  |
| <i>volume</i> | The volume of playback. |

#### 54.5.5.5 void DA7212\_Mute ( da7212\_handle\_t \* *handle*, bool *isMuted* )

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>handle</i>  | DA7212 handle pointer.               |
| <i>isMuted</i> | True means mute, false means unmute. |

#### 54.5.5.6 void DA7212\_ChangeInput ( da7212\_handle\_t \* *handle*, da7212\_Input\_t *DA7212\_Input* )

Parameters

|                     |                        |
|---------------------|------------------------|
| <i>handle</i>       | DA7212 handle pointer. |
| <i>DA7212_Input</i> | Input data source.     |

#### 54.5.5.7 void DA7212\_ChangeOutput ( da7212\_handle\_t \* *handle*, da7212\_Output\_t *DA7212\_Output* )

Parameters

|                      |                          |
|----------------------|--------------------------|
| <i>handle</i>        | DA7212 handle pointer.   |
| <i>DA7212_Output</i> | Output device of DA7212. |

#### 54.5.5.8 status\_t DA7212\_SetChannelVolume ( da7212\_handle\_t \* *handle*, uint32\_t *channel*, uint32\_t *volume* )

Parameters

---

|                |                                                     |
|----------------|-----------------------------------------------------|
| <i>handle</i>  | DA7212 handle pointer.                              |
| <i>channel</i> | shoule be a value of <code>_da7212_channel</code> . |
| <i>volume</i>  | volume range 0 - 0x3F mapped to range -57dB - 6dB.  |

#### 54.5.5.9 `status_t DA7212_SetChannelMute ( da7212_handle_t * handle, uint32_t channel, bool isMute )`

Parameters

|                |                                                     |
|----------------|-----------------------------------------------------|
| <i>handle</i>  | DA7212 handle pointer.                              |
| <i>channel</i> | shoule be a value of <code>_da7212_channel</code> . |
| <i>isMute</i>  | true is mute, false is unmute.                      |

#### 54.5.5.10 `status_t DA7212_SetProtocol ( da7212_handle_t * handle, da7212_protocol_t protocol )`

Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>handle</i>   | DA7212 handle pointer.           |
| <i>protocol</i> | <code>da7212_protocol_t</code> . |

#### 54.5.5.11 `status_t DA7212_SetMasterModeBits ( da7212_handle_t * handle, uint32_t bitWidth )`

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>handle</i>   | DA7212 handle pointer. |
| <i>bitWidth</i> | audio data bitwidth.   |

#### 54.5.5.12 `status_t DA7212_WriteRegister ( da7212_handle_t * handle, uint8_t u8Register, uint8_t u8RegisterData )`

## Parameters

|                       |                                        |
|-----------------------|----------------------------------------|
| <i>handle</i>         | DA7212 handle pointer.                 |
| <i>u8Register</i>     | DA7212 register address to be written. |
| <i>u8RegisterData</i> | Data to be written into register       |

**54.5.5.13** `status_t DA7212_ReadRegister ( da7212_handle_t * handle, uint8_t u8Register, uint8_t * pu8RegisterData )`

## Parameters

|                         |                                                |
|-------------------------|------------------------------------------------|
| <i>handle</i>           | DA7212 handle pointer.                         |
| <i>u8Register</i>       | DA7212 register address to be read.            |
| <i>pu8Register-Data</i> | Pointer where the read out value to be stored. |

**54.5.5.14** `status_t DA7212_Deinit ( da7212_handle_t * handle )`

## Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | DA7212 handle pointer. |
|---------------|------------------------|



## 54.5.6 DA7212 Adapter

### 54.5.6.1 Overview

The da7212 adapter provides a codec unify control interface.

#### Macros

- #define `HAL_CODEC_DA7212_HANDLER_SIZE` (`DA7212_I2C_HANDLER_SIZE + 4`)  
*codec handler size*

#### Functions

- `status_t HAL_CODEC_DA7212_Init` (void \*handle, void \*config)  
*Codec initialization.*
- `status_t HAL_CODEC_DA7212_Deinit` (void \*handle)  
*Codec de-initialization.*
- `status_t HAL_CODEC_DA7212_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- `status_t HAL_CODEC_DA7212_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- `status_t HAL_CODEC_DA7212_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- `status_t HAL_CODEC_DA7212_SetPower` (void \*handle, uint32\_t module, bool powerOn)  
*set audio codec module power.*
- `status_t HAL_CODEC_DA7212_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- `status_t HAL_CODEC_DA7212_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- `status_t HAL_CODEC_DA7212_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- `status_t HAL_CODEC_DA7212_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*
- static `status_t HAL_CODEC_Init` (void \*handle, void \*config)  
*Codec initialization.*
- static `status_t HAL_CODEC_Deinit` (void \*handle)  
*Codec de-initialization.*
- static `status_t HAL_CODEC_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- static `status_t HAL_CODEC_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- static `status_t HAL_CODEC_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- static `status_t HAL_CODEC_SetPower` (void \*handle, uint32\_t module, bool powerOn)

- set audio codec module power.*
- static `status_t HAL_CODEC_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- static `status_t HAL_CODEC_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- static `status_t HAL_CODEC_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- static `status_t HAL_CODEC_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

## 54.5.6.2 Function Documentation

### 54.5.6.2.1 `status_t HAL_CODEC_DA7212_Init ( void * handle, void * config )`

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

Returns

kStatus\_Success is success, else initial failed.

### 54.5.6.2.2 `status_t HAL_CODEC_DA7212_Deinit ( void * handle )`

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

### 54.5.6.2.3 `status_t HAL_CODEC_DA7212_SetFormat ( void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth )`

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.5.6.2.4 status\_t HAL\_CODEC\_DA7212\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

## Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.5.6.2.5 status\_t HAL\_CODEC\_DA7212\_SetMute ( void \* *handle*, uint32\_t *playChannel*, bool *isMute* )

## Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.5.6.2.6 status\_t HAL\_CODEC\_DA7212\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* )

## Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.5.6.2.7 status\_t HAL\_CODEC\_DA7212\_SetRecord ( void \* *handle*, uint32\_t *recordSource* )

## Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.5.6.2.8 status\_t HAL\_CODEC\_DA7212\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* )

## Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.5.6.2.9 status\_t HAL\_CODEC\_DA7212\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

## Parameters

|                   |                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                                 |
| <i>playSource</i> | audio codec play source, can be a value or combine value of <code>_codec_play_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

#### 54.5.6.2.10 `status_t HAL_CODEC_DA7212_ModuleControl ( void * handle, uint32_t cmd, uint32_t data )`

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

## Parameters

|               |                                                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                                                     |
| <i>cmd</i>    | module control cmd, reference <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.

#### 54.5.6.2.11 `static status_t HAL_CODEC_Init ( void * handle, void * config ) [inline], [static]`

## Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

## Returns

`kStatus_Success` is success, else initial failed.

#### 54.5.6.2.12 `static status_t HAL_CODEC_Deinit ( void * handle ) [inline], [static]`

## Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

## Returns

kStatus\_Success is success, else de-initial failed.

**54.5.6.2.13** `static status_t HAL_CODEC_SetFormat ( void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth ) [inline], [static]`

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

## Returns

kStatus\_Success is success, else configure failed.

**54.5.6.2.14** `static status_t HAL_CODEC_SetVolume ( void * handle, uint32_t playChannel, uint32_t volume ) [inline], [static]`

## Parameters

|                    |                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                                   |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of <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.

**54.5.6.2.15** `static status_t HAL_CODEC_SetMute ( void * handle, uint32_t playChannel, bool isMute ) [inline], [static]`

## Parameters

|                    |                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                                   |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> . |
| <i>isMute</i>      | true is mute, false is unmute.                                                                  |

## Returns

`kStatus_Success` is success, else configure failed.

**54.5.6.2.16** `static status_t HAL_CODEC_SetPower ( void * handle, uint32_t module, bool powerOn ) [inline], [static]`

## Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

## Returns

`kStatus_Success` is success, else configure failed.

**54.5.6.2.17** `static status_t HAL_CODEC_SetRecord ( void * handle, uint32_t recordSource ) [inline], [static]`

## Parameters

|                     |                                                                                                   |
|---------------------|---------------------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                                     |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of <code>_codec_record_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

**54.5.6.2.18** `static status_t HAL_CODEC_SetRecordChannel ( void * handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel ) [inline], [static]`

## Parameters

|                            |                                                                                                                                                              |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                                                |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference <code>_codec_record_channel</code> , can be a value or combine value of member in <code>_codec_record_channel</code> . |
| <i>rightRecord-Channel</i> | audio codec record channel, reference <code>_codec_record_channel</code> , can be a value or combine of member in <code>_codec_record_channel</code> .       |

## Returns

`kStatus_Success` is success, else configure failed.

**54.5.6.2.19** `static status_t HAL_CODEC_SetPlay ( void * handle, uint32_t playSource ) [inline], [static]`

## Parameters

|                   |                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                                 |
| <i>playSource</i> | audio codec play source, can be a value or combine value of <code>_codec_play_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

**54.5.6.2.20** `static status_t HAL_CODEC_ModuleControl ( void * handle, uint32_t cmd, uint32_t data ) [inline], [static]`

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

## Parameters

|               |                                                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                                                     |
| <i>cmd</i>    | module control cmd, reference <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.



## 54.6 SGTL5000 Driver

### 54.6.1 Overview

The sgtl5000 driver provides a codec control interface.

### Data Structures

- struct `sgtl_audio_format_t`  
*Audio format configuration. [More...](#)*
- struct `sgtl_config_t`  
*Initialize structure of sgtl5000. [More...](#)*
- struct `sgtl_handle_t`  
*SGTL codec handler. [More...](#)*

### Macros

- #define `CHIP_ID` 0x0000U  
*Define the register address of sgtl5000.*
- #define `SGTL5000_HEADPHONE_MAX_VOLUME_VALUE` 0x7FU  
*SGTL5000 volume setting range.*
- #define `SGTL5000_I2C_ADDR` 0x0A  
*SGTL5000 I2C address.*
- #define `SGTL_I2C_HANDLER_SIZE` `CODEC_I2C_MASTER_HANDLER_SIZE`  
*sgtl handle size*
- #define `SGTL_I2C_BITRATE` 100000U  
*sgtl i2c baudrate*

### Enumerations

- enum `sgtl_module_t` {  
  `kSGTL_ModuleADC` = 0x0,  
  `kSGTL_ModuleDAC`,  
  `kSGTL_ModuleDAP`,  
  `kSGTL_ModuleHP`,  
  `kSGTL_ModuleI2SIN`,  
  `kSGTL_ModuleI2SOUT`,  
  `kSGTL_ModuleLineIn`,  
  `kSGTL_ModuleLineOut`,  
  `kSGTL_ModuleMicin` }  
*Modules in Sgtl5000 board.*
- enum `sgtl_route_t` {

- ```

kSGTL_RouteBypass = 0x0,
kSGTL_RoutePlayback,
kSGTL_RoutePlaybackandRecord,
kSGTL_RoutePlaybackwithDAP,
kSGTL_RoutePlaybackwithDAPandRecord,
kSGTL_RouteRecord }
    Sgtl5000 data route.
• enum sgtl_protocol_t {
kSGTL_BusI2S = 0x0,
kSGTL_BusLeftJustified,
kSGTL_BusRightJustified,
kSGTL_BusPCMA,
kSGTL_BusPCMB }
    The audio data transfer protocol choice.
• enum {
kSGTL_HeadphoneLeft = 0,
kSGTL_HeadphoneRight = 1,
kSGTL_LineoutLeft = 2,
kSGTL_LineoutRight = 3 }
    sgtl play channel
• enum {
kSGTL_RecordSourceLineIn = 0U,
kSGTL_RecordSourceMic = 1U }
    sgtl record source _sgtl_record_source
• enum {
kSGTL_PlaySourceLineIn = 0U,
kSGTL_PlaySourceDAC = 1U }
    sgtl play source _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, 1)`)
CLOCK driver version 2.1.1.

54.6.2 Data Structure Documentation

54.6.2.1 struct sgtl_audio_format_t

Data Fields

- `uint32_t mclk_HZ`
master clock
- `uint32_t sampleRate`
Sample rate.
- `uint32_t bitWidth`
Bit width.
- `sgtl_sclk_edge_t sclkEdge`
sclk valid edge

54.6.2.2 struct sgtl_config_t

Data Fields

- `sgtl_route_t route`

- *Audio data route.*
sgtl_protocol_t bus
- *Audio transfer protocol.*
bool master_slave
- *Master or slave.*
sgtl_audio_format_t format
- *audio format*
uint8_t slaveAddress
- *code device slave address*
codec_i2c_config_t i2cConfig
- *i2c bus configuration*

Field Documentation

(1) sgtl_route_t sgtl_config_t::route

(2) bool sgtl_config_t::master_slave

True means master, false means slave.

54.6.2.3 struct sgtl_handle_t

Data Fields

- sgtl_config_t * config
sgtl config pointer
- uint8_t i2cHandle [SGTL_I2C_HANDLER_SIZE]
i2c handle

54.6.3 Macro Definition Documentation

54.6.3.1 #define FSL_SGTL5000_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

54.6.3.2 #define CHIP_ID 0x0000U

54.6.3.3 #define SGTL5000_I2C_ADDR 0x0A

54.6.4 Enumeration Type Documentation

54.6.4.1 enum sgtl_module_t

Enumerator

- kSGTL_ModuleADC* ADC module in SGTL5000.
- kSGTL_ModuleDAC* DAC module in SGTL5000.
- kSGTL_ModuleDAP* DAP module in SGTL5000.
- kSGTL_ModuleHP* Headphone module in SGTL5000.

kSGTL_ModuleI2SIN I2S-IN module in SGTL5000.
kSGTL_ModuleI2SOUT I2S-OUT module in SGTL5000.
kSGTL_ModuleLineIn Line-in module in SGTL5000.
kSGTL_ModuleLineOut Line-out module in SGTL5000.
kSGTL_ModuleMicin Micphone module in SGTL5000.

54.6.4.2 enum sgtl_route_t

Note

Only provide some typical data route, not all route listed. Users cannot combine any routes, once a new route is set, the previous one would be replaced.

Enumerator

kSGTL_RouteBypass LINEIN->Headphone.
kSGTL_RoutePlayback I2SIN->DAC->Headphone.
kSGTL_RoutePlaybackandRecord I2SIN->DAC->Headphone, LINEIN->ADC->I2SOUT.
kSGTL_RoutePlaybackwithDAP I2SIN->DAP->DAC->Headphone.
kSGTL_RoutePlaybackwithDAPandRecord I2SIN->DAP->DAC->HP, LINEIN->ADC->I2SOUT.
kSGTL_RouteRecord LINEIN->ADC->I2SOUT.

54.6.4.3 enum sgtl_protocol_t

Sgtl5000 only supports I2S format and PCM format.

Enumerator

kSGTL_BusI2S I2S Type.
kSGTL_BusLeftJustified Left justified.
kSGTL_BusRightJustified Right Justified.
kSGTL_BusPCMA PCMA.
kSGTL_BusPCMB PCMB.

54.6.4.4 anonymous enum

Enumerator

kSGTL_HeadphoneLeft headphone left channel
kSGTL_HeadphoneRight headphone right channel
kSGTL_LineoutLeft lineout left channel
kSGTL_LineoutRight lineout right channel

54.6.4.5 anonymous enum

Enumerator

kSGTL_RecordSourceLineIn record source line in
kSGTL_RecordSourceMic record source single end

54.6.4.6 anonymous enum

Enumerator

kSGTL_PlaySourceLineIn play source line in
kSGTL_PlaySourceDAC play source line in

54.6.4.7 enum sgtl_sclk_edge_t

Enumerator

kSGTL_SclkValidEdgeRising SCLK valid edge.
kSGTL_SclkValidEdgeFalling SCLK falling edge.

54.6.5 Function Documentation

54.6.5.1 status_t SGTL_Init (sgtl_handle_t * *handle*, sgtl_config_t * *config*)

This function calls SGTL_I2CInit(), and in this function, some configurations are fixed. The second parameter can be NULL. If users want to change the SGTL5000 settings, a configure structure should be prepared.

Note

If the codec_config is NULL, it would initialize sgtl5000 using default settings. The default setting:

```
* sgtl_init_t codec_config
* codec_config.route = kSGTL_RoutePlaybackandRecord
* codec_config.bus = kSGTL_BusI2S
* codec_config.master = slave
*
```

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>config</i>	sgtl5000 configuration structure. If this pointer equals to NULL, it means using the default configuration.

Returns

Initialization status

54.6.5.2 **status_t SGTL_SetDataRoute (sgtl_handle_t * *handle*, sgtl_route_t *route*)**

This function would set the data route according to route. The route cannot be combined, as all route would enable different modules.

Note

If a new route is set, the previous route would not work.

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>route</i>	Audio data route in sgtl5000.

54.6.5.3 **status_t SGTL_SetProtocol (sgtl_handle_t * *handle*, sgtl_protocol_t *protocol*)**

Sgtl5000 only supports I2S, I2S left, I2S right, PCM A, PCM B format.

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>protocol</i>	Audio data transfer protocol.

54.6.5.4 **void SGTL_SetMasterSlave (sgtl_handle_t * *handle*, bool *master*)**

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>master</i>	1 represent master, 0 represent slave.

54.6.5.5 **status_t SGTL_SetVolume (sgtl_handle_t * *handle*, sgtl_module_t *module*, uint32_t *volume*)**

This function would set the volume of sgtl5000 modules. This interface set module volume. The function assume that left channel and right channel has the same volume.

kSGTL_ModuleADC volume range: 0 - 0xF, 0dB - 22.5dB kSGTL_ModuleDAC volume range: 0x3C - 0xF0, 0dB - -90dB kSGTL_ModuleHP volume range: 0 - 0x7F, 12dB - -51.5dB kSGTL_ModuleLineOut volume range: 0 - 0x1F, 0.5dB steps

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>module</i>	Sgtl5000 module, such as DAC, ADC and etc.
<i>volume</i>	Volume value need to be set. The value is the exact value in register.

54.6.5.6 **uint32_t SGTL_GetVolume (sgtl_handle_t * *handle*, sgtl_module_t *module*)**

This function gets the volume of sgtl5000 modules. This interface get DAC module volume. The function assume that left channel and right channel has the same volume.

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>module</i>	Sgtl5000 module, such as DAC, ADC and etc.

Returns

Module value, the value is exact value in register.

54.6.5.7 **status_t SGTL_SetMute (sgtl_handle_t * *handle*, sgtl_module_t *module*, bool *mute*)**

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>module</i>	Sgtl5000 module, such as DAC, ADC and etc.
<i>mute</i>	True means mute, and false means unmute.

54.6.5.8 status_t SGTL_EnableModule (sgtl_handle_t * *handle*, sgtl_module_t *module*)

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>module</i>	Module expected to enable.

54.6.5.9 status_t SGTL_DisableModule (sgtl_handle_t * *handle*, sgtl_module_t *module*)

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>module</i>	Module expected to enable.

54.6.5.10 status_t SGTL_Deinit (sgtl_handle_t * *handle*)

Shut down Sgtl5000 modules.

Parameters

<i>handle</i>	Sgtl5000 handle structure pointer.
---------------	------------------------------------

54.6.5.11 status_t SGTL_ConfigDataFormat (sgtl_handle_t * *handle*, uint32_t *mclk*, uint32_t *sample_rate*, uint32_t *bits*)

This function would configure the registers about the sample rate, bit depths.

Parameters

<i>handle</i>	Sgtl5000 handle structure pointer.
<i>mclk</i>	Master clock frequency of I2S.
<i>sample_rate</i>	Sample rate of audio file running in sgtl5000. Sctl5000 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 (Sctl5000 only supports 16bit, 20bit, 24bit and 32 bit in HW).

54.6.5.12 `status_t SGTL_SetPlay (sctl_handle_t * handle, uint32_t playSource)`

Parameters

<i>handle</i>	Sctl5000 handle structure pointer.
<i>playSource</i>	play source value, reference <code>_sctl_play_source</code> .

Returns

`kStatus_Success`, else failed.

54.6.5.13 `status_t SGTL_SetRecord (sctl_handle_t * handle, uint32_t recordSource)`

Parameters

<i>handle</i>	Sctl5000 handle structure pointer.
<i>recordSource</i>	record source value, reference <code>_sctl_record_source</code> .

Returns

`kStatus_Success`, else failed.

54.6.5.14 `status_t SGTL_WriteReg (sctl_handle_t * handle, uint16_t reg, uint16_t val)`

Parameters

<i>handle</i>	Sctl5000 handle structure.
---------------	----------------------------

<i>reg</i>	The register address in sgtl.
<i>val</i>	Value needs to write into the register.

54.6.5.15 `status_t SGTL_ReadReg (sgtl_handle_t * handle, uint16_t reg, uint16_t * val)`

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>reg</i>	The register address in sgtl.
<i>val</i>	Value written to.

54.6.5.16 `status_t SGTL_ModifyReg (sgtl_handle_t * handle, uint16_t reg, uint16_t clr_mask, uint16_t val)`

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>reg</i>	The register address in sgtl.
<i>clr_mask</i>	The mask code for the bits want to write. The bit you want to write should be 0.
<i>val</i>	Value needs to write into the register.

54.6.6 SGTL5000 Adapter

54.6.6.1 Overview

The sgtl5000 adapter provides a codec unify control interface.

Macros

- #define `HAL_CODEC_SGTL_HANDLER_SIZE` (`SGTL_I2C_HANDLER_SIZE + 4`)
codec handler size

Functions

- `status_t HAL_CODEC_SGTL5000_Init` (void *handle, void *config)
Codec initialization.
- `status_t HAL_CODEC_SGTL5000_Deinit` (void *handle)
Codec de-initialization.
- `status_t HAL_CODEC_SGTL5000_SetFormat` (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
set audio data format.
- `status_t HAL_CODEC_SGTL5000_SetVolume` (void *handle, uint32_t playChannel, uint32_t volume)
set audio codec module volume.
- `status_t HAL_CODEC_SGTL5000_SetMute` (void *handle, uint32_t playChannel, bool isMute)
set audio codec module mute.
- `status_t HAL_CODEC_SGTL5000_SetPower` (void *handle, uint32_t module, bool powerOn)
set audio codec module power.
- `status_t HAL_CODEC_SGTL5000_SetRecord` (void *handle, uint32_t recordSource)
codec set record source.
- `status_t HAL_CODEC_SGTL5000_SetRecordChannel` (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
codec set record channel.
- `status_t HAL_CODEC_SGTL5000_SetPlay` (void *handle, uint32_t playSource)
codec set play source.
- `status_t HAL_CODEC_SGTL5000_ModuleControl` (void *handle, uint32_t cmd, uint32_t data)
codec module control.
- static `status_t HAL_CODEC_Init` (void *handle, void *config)
Codec initialization.
- static `status_t HAL_CODEC_Deinit` (void *handle)
Codec de-initialization.
- static `status_t HAL_CODEC_SetFormat` (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
set audio data format.
- static `status_t HAL_CODEC_SetVolume` (void *handle, uint32_t playChannel, uint32_t volume)
set audio codec module volume.
- static `status_t HAL_CODEC_SetMute` (void *handle, uint32_t playChannel, bool isMute)
set audio codec module mute.
- static `status_t HAL_CODEC_SetPower` (void *handle, uint32_t module, bool powerOn)

- *set audio codec module power.*
static [status_t HAL_CODEC_SetRecord](#) (void *handle, uint32_t recordSource)
codec set record source.
- static [status_t HAL_CODEC_SetRecordChannel](#) (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
codec set record channel.
- static [status_t HAL_CODEC_SetPlay](#) (void *handle, uint32_t playSource)
codec set play source.
- static [status_t HAL_CODEC_ModuleControl](#) (void *handle, uint32_t cmd, uint32_t data)
codec module control.

54.6.6.2 Function Documentation

54.6.6.2.1 status_t HAL_CODEC_SGTL5000_Init (void * handle, void * config)

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configuration.

Returns

kStatus_Success is success, else initial failed.

54.6.6.2.2 status_t HAL_CODEC_SGTL5000_Deinit (void * handle)

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus_Success is success, else de-initial failed.

54.6.6.2.3 status_t HAL_CODEC_SGTL5000_SetFormat (void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus_Success is success, else configure failed.

54.6.6.2.4 status_t HAL_CODEC_SGTL5000_SetVolume (void * *handle*, uint32_t *playChannel*, uint32_t *volume*)

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

54.6.6.2.5 status_t HAL_CODEC_SGTL5000_SetMute (void * *handle*, uint32_t *playChannel*, bool *isMute*)

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>isMute</i>	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

54.6.6.2.6 status_t HAL_CODEC_SGTL5000_SetPower (void * *handle*, uint32_t *module*, bool *powerOn*)

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

54.6.6.2.7 status_t HAL_CODEC_SGTL5000_SetRecord (void * *handle*, uint32_t *recordSource*)

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

54.6.6.2.8 status_t HAL_CODEC_SGTL5000_SetRecordChannel (void * *handle*, uint32_t *leftRecordChannel*, uint32_t *rightRecordChannel*)

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

54.6.6.2.9 status_t HAL_CODEC_SGTL5000_SetPlay (void * *handle*, uint32_t *playSource*)

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of <code>_codec_play_source</code> .

Returns

`kStatus_Success` is success, else configure failed.

54.6.6.2.10 `status_t HAL_CODEC_SGTL5000_ModuleControl (void * handle, uint32_t cmd, uint32_t data)`

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference <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.

54.6.6.2.11 `static status_t HAL_CODEC_Init (void * handle, void * config) [inline], [static]`

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configuration.

Returns

`kStatus_Success` is success, else initial failed.

54.6.6.2.12 `static status_t HAL_CODEC_Deinit (void * handle) [inline], [static]`

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus_Success is success, else de-initial failed.

54.6.6.2.13 `static status_t HAL_CODEC_SetFormat (void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth) [inline], [static]`

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus_Success is success, else configure failed.

54.6.6.2.14 `static status_t HAL_CODEC_SetVolume (void * handle, uint32_t playChannel, uint32_t volume) [inline], [static]`

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of <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.

54.6.6.2.15 `static status_t HAL_CODEC_SetMute (void * handle, uint32_t playChannel, bool isMute) [inline], [static]`

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> .
<i>isMute</i>	true is mute, false is unmute.

Returns

`kStatus_Success` is success, else configure failed.

54.6.6.2.16 `static status_t HAL_CODEC_SetPower (void * handle, uint32_t module, bool powerOn) [inline], [static]`

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

`kStatus_Success` is success, else configure failed.

54.6.6.2.17 `static status_t HAL_CODEC_SetRecord (void * handle, uint32_t recordSource) [inline], [static]`

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of <code>_codec_record_source</code> .

Returns

`kStatus_Success` is success, else configure failed.

54.6.6.2.18 `static status_t HAL_CODEC_SetRecordChannel (void * handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel) [inline], [static]`

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference <code>_codec_record_channel</code> , can be a value or combine value of member in <code>_codec_record_channel</code> .
<i>rightRecord-Channel</i>	audio codec record channel, reference <code>_codec_record_channel</code> , can be a value or combine value of member in <code>_codec_record_channel</code> .

Returns

`kStatus_Success` is success, else configure failed.

54.6.6.2.19 `static status_t HAL_CODEC_SetPlay (void * handle, uint32_t playSource) [inline], [static]`

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of <code>_codec_play_source</code> .

Returns

`kStatus_Success` is success, else configure failed.

54.6.6.2.20 `static status_t HAL_CODEC_ModuleControl (void * handle, uint32_t cmd, uint32_t data) [inline], [static]`

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference <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.

54.7 WM8960 Driver

54.7.1 Overview

The wm8960 driver provides a codec control interface.

Data Structures

- struct [wm8960_audio_format_t](#)
wm8960 audio format [More...](#)
- struct [wm8960_master_sysclk_config_t](#)
wm8960 master system clock configuration [More...](#)
- struct [wm8960_config_t](#)
Initialize structure of WM8960. [More...](#)
- struct [wm8960_handle_t](#)
wm8960 codec handler [More...](#)

Macros

- #define [WM8960_I2C_HANDLER_SIZE](#) [CODEC_I2C_MASTER_HANDLER_SIZE](#)
wm8960 handle size
- #define [WM8960_LINVOL](#) 0x0U
Define the register address of WM8960.
- #define [WM8960_CACHEREGNUM](#) 56U
Cache register number.
- #define [WM8960_CLOCK2_BCLK_DIV_MASK](#) 0xFU
WM8960 CLOCK2 bits.
- #define [WM8960_IFACE1_FORMAT_MASK](#) 0x03U
WM8960_IFACE1 FORMAT bits.
- #define [WM8960_IFACE1_WL_MASK](#) 0x0CU
WM8960_IFACE1 WL bits.
- #define [WM8960_IFACE1_LRP_MASK](#) 0x10U
WM8960_IFACE1 LRP bit.
- #define [WM8960_IFACE1_DLRSWAP_MASK](#) 0x20U
WM8960_IFACE1 DLRSWAP bit.
- #define [WM8960_IFACE1_MS_MASK](#) 0x40U
WM8960_IFACE1 MS bit.
- #define [WM8960_IFACE1_BCLKINV_MASK](#) 0x80U
WM8960_IFACE1 BCLKINV bit.
- #define [WM8960_IFACE1_ALRSWAP_MASK](#) 0x100U
WM8960_IFACE1 ALRSWAP bit.
- #define [WM8960_POWER1_VREF_MASK](#) 0x40U
WM8960_POWER1.
- #define [WM8960_POWER2_DACL_MASK](#) 0x100U
WM8960_POWER2.
- #define [WM8960_I2C_ADDR](#) 0x1A
WM8960 I2C address.
- #define [WM8960_I2C_BAUDRATE](#) (100000U)

- *WM8960 I2C baudrate.*
- #define `WM8960_ADC_MAX_VOLUME_VALUE` `0xFFU`
WM8960 maximum volume value.

Enumerations

- enum `wm8960_module_t` {
`kWM8960_ModuleADC` = 0,
`kWM8960_ModuleDAC` = 1,
`kWM8960_ModuleVREF` = 2,
`kWM8960_ModuleHP` = 3,
`kWM8960_ModuleMICB` = 4,
`kWM8960_ModuleMIC` = 5,
`kWM8960_ModuleLineIn` = 6,
`kWM8960_ModuleLineOut` = 7,
`kWM8960_ModuleSpeaker` = 8,
`kWM8960_ModuleOMIX` = 9 }
Modules in WM8960 board.
- enum {
`kWM8960_HeadphoneLeft` = 1,
`kWM8960_HeadphoneRight` = 2,
`kWM8960_SpeakerLeft` = 4,
`kWM8960_SpeakerRight` = 8 }
wm8960 play channel
- enum `wm8960_play_source_t` {
`kWM8960_PlaySourcePGA` = 1,
`kWM8960_PlaySourceInput` = 2,
`kWM8960_PlaySourceDAC` = 4 }
wm8960 play source
- enum `wm8960_route_t` {
`kWM8960_RouteBypass` = 0,
`kWM8960_RoutePlayback` = 1,
`kWM8960_RoutePlaybackandRecord` = 2,
`kWM8960_RouteRecord` = 5 }
WM8960 data route.
- enum `wm8960_protocol_t` {
`kWM8960_BusI2S` = 2,
`kWM8960_BusLeftJustified` = 1,
`kWM8960_BusRightJustified` = 0,
`kWM8960_BusPCMA` = 3,
`kWM8960_BusPCMB` = 3 | (1 << 4) }
The audio data transfer protocol choice.
- enum `wm8960_input_t` {

- ```

kWM8960_InputClosed = 0,
kWM8960_InputSingleEndedMic = 1,
kWM8960_InputDifferentialMicInput2 = 2,
kWM8960_InputDifferentialMicInput3 = 3,
kWM8960_InputLineINPUT2 = 4,
kWM8960_InputLineINPUT3 = 5 }
 wm8960 input source
• enum {
kWM8960_AudioSampleRate8KHz = 8000U,
kWM8960_AudioSampleRate11025Hz = 11025U,
kWM8960_AudioSampleRate12KHz = 12000U,
kWM8960_AudioSampleRate16KHz = 16000U,
kWM8960_AudioSampleRate22050Hz = 22050U,
kWM8960_AudioSampleRate24KHz = 24000U,
kWM8960_AudioSampleRate32KHz = 32000U,
kWM8960_AudioSampleRate44100Hz = 44100U,
kWM8960_AudioSampleRate48KHz = 48000U,
kWM8960_AudioSampleRate96KHz = 96000U,
kWM8960_AudioSampleRate192KHz = 192000U,
kWM8960_AudioSampleRate384KHz = 384000U }
 audio sample rate definition
• enum {
kWM8960_AudioBitWidth16bit = 16U,
kWM8960_AudioBitWidth20bit = 20U,
kWM8960_AudioBitWidth24bit = 24U,
kWM8960_AudioBitWidth32bit = 32U }
 audio bit width
• enum wm8960_sysclk_source_t {
kWM8960_SysClkSourceMclk = 0U,
kWM8960_SysClkSourceInternalPLL = 1U }
 wm8960 sysclk source

```

## Functions

- `status_t WM8960_Init (wm8960_handle_t *handle, const wm8960_config_t *config)`  
*WM8960 initialize function.*
- `status_t WM8960_Deinit (wm8960_handle_t *handle)`  
*Deinit the WM8960 codec.*
- `status_t WM8960_SetDataRoute (wm8960_handle_t *handle, wm8960_route_t route)`  
*Set audio data route in WM8960.*
- `status_t WM8960_SetLeftInput (wm8960_handle_t *handle, wm8960_input_t input)`  
*Set left audio input source in WM8960.*
- `status_t WM8960_SetRightInput (wm8960_handle_t *handle, wm8960_input_t input)`  
*Set right audio input source in WM8960.*
- `status_t WM8960_SetProtocol (wm8960_handle_t *handle, wm8960_protocol_t protocol)`  
*Set the audio transfer protocol.*

- void `WM8960_SetMasterSlave` (`wm8960_handle_t *handle`, bool master)  
*Set WM8960 as master or slave.*
- `status_t WM8960_SetVolume` (`wm8960_handle_t *handle`, `wm8960_module_t module`, `uint32_t volume`)  
*Set the volume of different modules in WM8960.*
- `uint32_t WM8960_GetVolume` (`wm8960_handle_t *handle`, `wm8960_module_t module`)  
*Get the volume of different modules in WM8960.*
- `status_t WM8960_SetMute` (`wm8960_handle_t *handle`, `wm8960_module_t module`, bool isEnabled)  
*Mute modules in WM8960.*
- `status_t WM8960_SetModule` (`wm8960_handle_t *handle`, `wm8960_module_t module`, bool isEnabled)  
*Enable/disable expected devices.*
- `status_t WM8960_SetPlay` (`wm8960_handle_t *handle`, `uint32_t playSource`)  
*SET the WM8960 play source.*
- `status_t WM8960_ConfigDataFormat` (`wm8960_handle_t *handle`, `uint32_t sysclk`, `uint32_t sample_rate`, `uint32_t bits`)  
*Configure the data format of audio data.*
- `status_t WM8960_SetJackDetect` (`wm8960_handle_t *handle`, bool isEnabled)  
*Enable/disable jack detect feature.*
- `status_t WM8960_WriteReg` (`wm8960_handle_t *handle`, `uint8_t reg`, `uint16_t val`)  
*Write register to WM8960 using I2C.*
- `status_t WM8960_ReadReg` (`uint8_t reg`, `uint16_t *val`)  
*Read register from WM8960 using I2C.*
- `status_t WM8960_ModifyReg` (`wm8960_handle_t *handle`, `uint8_t reg`, `uint16_t mask`, `uint16_t val`)  
*Modify some bits in the register using I2C.*

## Driver version

- `#define FSL_WM8960_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))`  
*CLOCK driver version 2.2.2.*

## 54.7.2 Data Structure Documentation

### 54.7.2.1 struct `wm8960_audio_format_t`

#### Data Fields

- `uint32_t mclk_HZ`  
*master clock frequency*
- `uint32_t sampleRate`  
*sample rate*
- `uint32_t bitWidth`  
*bit width*

### 54.7.2.2 struct wm8960\_master\_sysclk\_config\_t

#### Data Fields

- [wm8960\\_sysclk\\_source\\_t sysclkSource](#)  
*sysclk source*
- [uint32\\_t sysclkFreq](#)  
*PLL output frequency value.*

### 54.7.2.3 struct wm8960\_config\_t

#### Data Fields

- [wm8960\\_route\\_t route](#)  
*Audio data route.*
- [wm8960\\_protocol\\_t bus](#)  
*Audio transfer protocol.*
- [wm8960\\_audio\\_format\\_t format](#)  
*Audio format.*
- [bool master\\_slave](#)  
*Master or slave.*
- [wm8960\\_master\\_sysclk\\_config\\_t masterClock](#)  
*master clock configurations*
- [bool enableSpeaker](#)  
*True means enable class D speaker as output, false means no.*
- [wm8960\\_input\\_t leftInputSource](#)  
*Left input source for WM8960.*
- [wm8960\\_input\\_t rightInputSource](#)  
*Right input source for wm8960.*
- [wm8960\\_play\\_source\\_t playSource](#)  
*play source*
- [uint8\\_t slaveAddress](#)  
*wm8960 device address*
- [codec\\_i2c\\_config\\_t i2cConfig](#)  
*i2c configuration*

#### Field Documentation

(1) [wm8960\\_route\\_t wm8960\\_config\\_t::route](#)

(2) [bool wm8960\\_config\\_t::master\\_slave](#)

### 54.7.2.4 struct wm8960\_handle\_t

#### Data Fields

- [const wm8960\\_config\\_t \\* config](#)  
*wm8904 config pointer*
- [uint8\\_t i2cHandle \[WM8960\\_I2C\\_HANDLER\\_SIZE\]](#)  
*i2c handle*



### 54.7.3 Macro Definition Documentation

54.7.3.1 `#define WM8960_LINVOL 0x0U`

54.7.3.2 `#define WM8960_I2C_ADDR 0x1A`

### 54.7.4 Enumeration Type Documentation

#### 54.7.4.1 `enum wm8960_module_t`

Enumerator

*kWM8960\_ModuleADC* ADC module in WM8960.

*kWM8960\_ModuleDAC* DAC module in WM8960.

*kWM8960\_ModuleVREF* VREF module.

*kWM8960\_ModuleHP* Headphone.

*kWM8960\_ModuleMICB* Mic bias.

*kWM8960\_ModuleMIC* Input Mic.

*kWM8960\_ModuleLineIn* Analog in PGA.

*kWM8960\_ModuleLineOut* Line out module.

*kWM8960\_ModuleSpeaker* Speaker module.

*kWM8960\_ModuleOMIX* Output mixer.

#### 54.7.4.2 anonymous enum

Enumerator

*kWM8960\_HeadphoneLeft* wm8960 headphone left channel

*kWM8960\_HeadphoneRight* wm8960 headphone right channel

*kWM8960\_SpeakerLeft* wm8960 speaker left channel

*kWM8960\_SpeakerRight* wm8960 speaker right channel

#### 54.7.4.3 `enum wm8960_play_source_t`

Enumerator

*kWM8960\_PlaySourcePGA* wm8960 play source PGA

*kWM8960\_PlaySourceInput* wm8960 play source Input

*kWM8960\_PlaySourceDAC* wm8960 play source DAC

#### 54.7.4.4 `enum wm8960_route_t`

Only provide some typical data route, not all route listed. Note: Users cannot combine any routes, once a new route is set, the previous one would be replaced.

Enumerator

*kWM8960\_RouteBypass* LINEIN->Headphone.  
*kWM8960\_RoutePlayback* I2SIN->DAC->Headphone.  
*kWM8960\_RoutePlaybackandRecord* I2SIN->DAC->Headphone, LINEIN->ADC->I2SOUT.  
*kWM8960\_RouteRecord* LINEIN->ADC->I2SOUT.

#### 54.7.4.5 enum wm8960\_protocol\_t

WM8960 only supports I2S format and PCM format.

Enumerator

*kWM8960\_BusI2S* I2S type.  
*kWM8960\_BusLeftJustified* Left justified mode.  
*kWM8960\_BusRightJustified* Right justified mode.  
*kWM8960\_BusPCMA* PCM A mode.  
*kWM8960\_BusPCMB* PCM B mode.

#### 54.7.4.6 enum wm8960\_input\_t

Enumerator

*kWM8960\_InputClosed* Input device is closed.  
*kWM8960\_InputSingleEndedMic* Input as single ended mic, only use L/RINPUT1.  
*kWM8960\_InputDifferentialMicInput2* Input as differential mic, use L/RINPUT1 and L/RINPUT2.  
*kWM8960\_InputDifferentialMicInput3* Input as differential mic, use L/RINPUT1 and L/RINPUT3.  
*kWM8960\_InputLineINPUT2* Input as line input, only use L/RINPUT2.  
*kWM8960\_InputLineINPUT3* Input as line input, only use L/RINPUT3.

#### 54.7.4.7 anonymous enum

Enumerator

*kWM8960\_AudioSampleRate8KHz* Sample rate 8000 Hz.  
*kWM8960\_AudioSampleRate11025Hz* Sample rate 11025 Hz.  
*kWM8960\_AudioSampleRate12KHz* Sample rate 12000 Hz.  
*kWM8960\_AudioSampleRate16KHz* Sample rate 16000 Hz.  
*kWM8960\_AudioSampleRate22050Hz* Sample rate 22050 Hz.  
*kWM8960\_AudioSampleRate24KHz* Sample rate 24000 Hz.  
*kWM8960\_AudioSampleRate32KHz* Sample rate 32000 Hz.  
*kWM8960\_AudioSampleRate44100Hz* Sample rate 44100 Hz.  
*kWM8960\_AudioSampleRate48KHz* Sample rate 48000 Hz.

*kWM8960\_AudioSampleRate96KHz* Sample rate 96000 Hz.  
*kWM8960\_AudioSampleRate192KHz* Sample rate 192000 Hz.  
*kWM8960\_AudioSampleRate384KHz* Sample rate 384000 Hz.

#### 54.7.4.8 anonymous enum

Enumerator

*kWM8960\_AudioBitWidth16bit* audio bit width 16  
*kWM8960\_AudioBitWidth20bit* audio bit width 20  
*kWM8960\_AudioBitWidth24bit* audio bit width 24  
*kWM8960\_AudioBitWidth32bit* audio bit width 32

#### 54.7.4.9 enum wm8960\_sysclk\_source\_t

Enumerator

*kWM8960\_SysClkSourceMclk* sysclk source from external MCLK  
*kWM8960\_SysClkSourceInternalPLL* sysclk source from internal PLL

### 54.7.5 Function Documentation

#### 54.7.5.1 `status_t WM8960_Init ( wm8960_handle_t * handle, const wm8960_config_t * config )`

The second parameter is NULL to WM8960 in this version. If users want to change the settings, they have to use `wm8960_write_reg()` or `wm8960_modify_reg()` to set the register value of WM8960. Note: If the `codec_config` is NULL, it would initialize WM8960 using default settings. The default setting: `codec_config->route = kWM8960_RoutePlaybackandRecord` `codec_config->bus = kWM8960_BusI2S` `codec_config->master = slave`

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>handle</i> | WM8960 handle structure.        |
| <i>config</i> | WM8960 configuration structure. |

#### 54.7.5.2 `status_t WM8960_Deinit ( wm8960_handle_t * handle )`

This function close all modules in WM8960 to save power.

## Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>handle</i> | WM8960 handle structure pointer. |
|---------------|----------------------------------|

#### 54.7.5.3 **status\_t WM8960\_SetDataRoute ( wm8960\_handle\_t \* *handle*, wm8960\_route\_t *route* )**

This function would set the data route according to route. The route cannot be combined, as all route would enable different modules. Note: If a new route is set, the previous route would not work.

## Parameters

|               |                             |
|---------------|-----------------------------|
| <i>handle</i> | WM8960 handle structure.    |
| <i>route</i>  | Audio data route in WM8960. |

#### 54.7.5.4 **status\_t WM8960\_SetLeftInput ( wm8960\_handle\_t \* *handle*, wm8960\_input\_t *input* )**

## Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8960 handle structure. |
| <i>input</i>  | Audio input source.      |

#### 54.7.5.5 **status\_t WM8960\_SetRightInput ( wm8960\_handle\_t \* *handle*, wm8960\_input\_t *input* )**

## Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8960 handle structure. |
| <i>input</i>  | Audio input source.      |

#### 54.7.5.6 **status\_t WM8960\_SetProtocol ( wm8960\_handle\_t \* *handle*, wm8960\_protocol\_t *protocol* )**

WM8960 only supports I2S, left justified, right justified, PCM A, PCM B format.

Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>handle</i>   | WM8960 handle structure.      |
| <i>protocol</i> | Audio data transfer protocol. |

#### 54.7.5.7 void WM8960\_SetMasterSlave ( wm8960\_handle\_t \* handle, bool master )

Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>handle</i> | WM8960 handle structure.               |
| <i>master</i> | 1 represent master, 0 represent slave. |

#### 54.7.5.8 status\_t WM8960\_SetVolume ( wm8960\_handle\_t \* handle, wm8960\_module\_t module, uint32\_t volume )

This function would set the volume of WM8960 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Module:kWM8960\_ModuleADC, volume range value: 0 is mute, 1-255 is -97db to 30db  
 Module:kWM8960\_ModuleDAC, volume range value: 0 is mute, 1-255 is -127db to 0db  
 Module:kWM8960\_ModuleHP, volume range value: 0 - 2F is mute, 0x30 - 0x7F is -73db to 6db  
 Module:kWM8960\_ModuleLineIn, volume range value: 0 - 0x3F is -17.25db to 30db  
 Module:kWM8960\_ModuleSpeaker, volume range value: 0 - 2F is mute, 0x30 - 0x7F is -73db to 6db

Parameters

|               |                                                                |
|---------------|----------------------------------------------------------------|
| <i>handle</i> | WM8960 handle structure.                                       |
| <i>module</i> | Module to set volume, it can be ADC, DAC, Headphone and so on. |
| <i>volume</i> | Volume value need to be set.                                   |

#### 54.7.5.9 uint32\_t WM8960\_GetVolume ( wm8960\_handle\_t \* handle, wm8960\_module\_t module )

This function gets the volume of WM8960 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

## Parameters

|               |                                                                |
|---------------|----------------------------------------------------------------|
| <i>handle</i> | WM8960 handle structure.                                       |
| <i>module</i> | Module to set volume, it can be ADC, DAC, Headphone and so on. |

## Returns

Volume value of the module.

#### 54.7.5.10 **status\_t WM8960\_SetMute ( wm8960\_handle\_t \* *handle*, wm8960\_module\_t *module*, bool *isEnabled* )**

## Parameters

|                  |                                   |
|------------------|-----------------------------------|
| <i>handle</i>    | WM8960 handle structure.          |
| <i>module</i>    | Modules need to be mute.          |
| <i>isEnabled</i> | Mute or unmute, 1 represent mute. |

#### 54.7.5.11 **status\_t WM8960\_SetModule ( wm8960\_handle\_t \* *handle*, wm8960\_module\_t *module*, bool *isEnabled* )**

## Parameters

|                  |                            |
|------------------|----------------------------|
| <i>handle</i>    | WM8960 handle structure.   |
| <i>module</i>    | Module expected to enable. |
| <i>isEnabled</i> | Enable or disable moudles. |

#### 54.7.5.12 **status\_t WM8960\_SetPlay ( wm8960\_handle\_t \* *handle*, uint32\_t *playSource* )**

## Parameters

|                   |                                                                                                                                                                                                     |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>     | WM8960 handle structure.                                                                                                                                                                            |
| <i>playSource</i> | play source , can be a value combine of kWM8960_ModuleHeadphoneSourcePGA, kWM8960_ModuleHeadphoneSourceDAC, kWM8960_ModulePlaySourceInput, kWM8960_ModulePlayMonoRight, kWM8960_ModulePlayMonoLeft. |

## Returns

kStatus\_WM8904\_Success if successful, different code otherwise..

**54.7.5.13** `status_t WM8960_ConfigDataFormat ( wm8960_handle_t * handle, uint32_t sysclk, uint32_t sample_rate, uint32_t bits )`

This function would configure the registers about the sample rate, bit depths.

## Parameters

|                    |                                                                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>      | WM8960 handle structure pointer.                                                                                                          |
| <i>sysclk</i>      | system clock of the codec which can be generated by MCLK or PLL output.                                                                   |
| <i>sample_rate</i> | Sample rate of audio file running in WM8960. WM8960 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate. |
| <i>bits</i>        | Bit depth of audio file (WM8960 only supports 16bit, 20bit, 24bit and 32 bit in HW).                                                      |

**54.7.5.14** `status_t WM8960_SetJackDetect ( wm8960_handle_t * handle, bool isEnabled )`

## Parameters

|                  |                            |
|------------------|----------------------------|
| <i>handle</i>    | WM8960 handle structure.   |
| <i>isEnabled</i> | Enable or disable moudles. |

**54.7.5.15** `status_t WM8960_WriteReg ( wm8960_handle_t * handle, uint8_t reg, uint16_t val )`

## Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>handle</i> | WM8960 handle structure.                |
| <i>reg</i>    | The register address in WM8960.         |
| <i>val</i>    | Value needs to write into the register. |

**54.7.5.16** `status_t WM8960_ReadReg ( uint8_t reg, uint16_t * val )`

## Parameters

|            |                                 |
|------------|---------------------------------|
| <i>reg</i> | The register address in WM8960. |
| <i>val</i> | Value written to.               |

**54.7.5.17** `status_t WM8960_ModifyReg ( wm8960_handle_t * handle, uint8_t reg, uint16_t mask, uint16_t val )`



## Parameters

|               |                                                                                  |
|---------------|----------------------------------------------------------------------------------|
| <i>handle</i> | WM8960 handle structure.                                                         |
| <i>reg</i>    | The register address in WM8960.                                                  |
| <i>mask</i>   | The mask code for the bits want to write. The bit you want to write should be 0. |
| <i>val</i>    | Value needs to write into the register.                                          |

## 54.7.6 WM8960 Adapter

### 54.7.6.1 Overview

The wm8960 adapter provides a codec unify control interface.

#### Macros

- #define `HAL_CODEC_WM8960_HANDLER_SIZE` (`WM8960_I2C_HANDLER_SIZE + 4`)  
*codec handler size*

#### Functions

- `status_t HAL_CODEC_WM8960_Init` (void \*handle, void \*config)  
*Codec initialization.*
- `status_t HAL_CODEC_WM8960_Deinit` (void \*handle)  
*Codec de-initialization.*
- `status_t HAL_CODEC_WM8960_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- `status_t HAL_CODEC_WM8960_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- `status_t HAL_CODEC_WM8960_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- `status_t HAL_CODEC_WM8960_SetPower` (void \*handle, uint32\_t module, bool powerOn)  
*set audio codec module power.*
- `status_t HAL_CODEC_WM8960_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- `status_t HAL_CODEC_WM8960_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- `status_t HAL_CODEC_WM8960_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- `status_t HAL_CODEC_WM8960_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*
- static `status_t HAL_CODEC_Init` (void \*handle, void \*config)  
*Codec initialization.*
- static `status_t HAL_CODEC_Deinit` (void \*handle)  
*Codec de-initialization.*
- static `status_t HAL_CODEC_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- static `status_t HAL_CODEC_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- static `status_t HAL_CODEC_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- static `status_t HAL_CODEC_SetPower` (void \*handle, uint32\_t module, bool powerOn)

- *set audio codec module power.*  
static [status\\_t HAL\\_CODEC\\_SetRecord](#) (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- static [status\\_t HAL\\_CODEC\\_SetRecordChannel](#) (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- static [status\\_t HAL\\_CODEC\\_SetPlay](#) (void \*handle, uint32\_t playSource)  
*codec set play source.*
- static [status\\_t HAL\\_CODEC\\_ModuleControl](#) (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

## 54.7.6.2 Function Documentation

### 54.7.6.2.1 status\_t HAL\_CODEC\_WM8960\_Init ( void \* handle, void \* config )

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

Returns

kStatus\_Success is success, else initial failed.

### 54.7.6.2.2 status\_t HAL\_CODEC\_WM8960\_Deinit ( void \* handle )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

### 54.7.6.2.3 status\_t HAL\_CODEC\_WM8960\_SetFormat ( void \* handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth )

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.7.6.2.4 status\_t HAL\_CODEC\_WM8960\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

## Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.7.6.2.5 status\_t HAL\_CODEC\_WM8960\_SetMute ( void \* *handle*, uint32\_t *playChannel*, bool *isMute* )

## Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.7.6.2.6 status\_t HAL\_CODEC\_WM8960\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* )

## Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.7.6.2.7 status\_t HAL\_CODEC\_WM8960\_SetRecord ( void \* *handle*, uint32\_t *recordSource* )

## Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.7.6.2.8 status\_t HAL\_CODEC\_WM8960\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* )

## Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.7.6.2.9 status\_t HAL\_CODEC\_WM8960\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

## Parameters

|                   |                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                                 |
| <i>playSource</i> | audio codec play source, can be a value or combine value of <code>_codec_play_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

#### 54.7.6.2.10 `status_t HAL_CODEC_WM8960_ModuleControl ( void * handle, uint32_t cmd, uint32_t data )`

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

## Parameters

|               |                                                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                                                     |
| <i>cmd</i>    | module control cmd, reference <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.

#### 54.7.6.2.11 `static status_t HAL_CODEC_Init ( void * handle, void * config ) [inline], [static]`

## Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

## Returns

`kStatus_Success` is success, else initial failed.

#### 54.7.6.2.12 `static status_t HAL_CODEC_Deinit ( void * handle ) [inline], [static]`

## Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

## Returns

kStatus\_Success is success, else de-initial failed.

**54.7.6.2.13** `static status_t HAL_CODEC_SetFormat ( void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth ) [inline], [static]`

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

## Returns

kStatus\_Success is success, else configure failed.

**54.7.6.2.14** `static status_t HAL_CODEC_SetVolume ( void * handle, uint32_t playChannel, uint32_t volume ) [inline], [static]`

## Parameters

|                    |                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                                   |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of <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.

**54.7.6.2.15** `static status_t HAL_CODEC_SetMute ( void * handle, uint32_t playChannel, bool isMute ) [inline], [static]`

## Parameters

|                    |                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                                   |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> . |
| <i>isMute</i>      | true is mute, false is unmute.                                                                  |

## Returns

`kStatus_Success` is success, else configure failed.

**54.7.6.2.16** `static status_t HAL_CODEC_SetPower ( void * handle, uint32_t module, bool powerOn ) [inline], [static]`

## Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

## Returns

`kStatus_Success` is success, else configure failed.

**54.7.6.2.17** `static status_t HAL_CODEC_SetRecord ( void * handle, uint32_t recordSource ) [inline], [static]`

## Parameters

|                     |                                                                                                   |
|---------------------|---------------------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                                     |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of <code>_codec_record_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

**54.7.6.2.18** `static status_t HAL_CODEC_SetRecordChannel ( void * handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel ) [inline], [static]`



## Parameters

|                            |                                                                                                                                                              |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                                                |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference <code>_codec_record_channel</code> , can be a value or combine value of member in <code>_codec_record_channel</code> . |
| <i>rightRecord-Channel</i> | audio codec record channel, reference <code>_codec_record_channel</code> , can be a value or combine value of member in <code>_codec_record_channel</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

**54.7.6.2.19** `static status_t HAL_CODEC_SetPlay ( void * handle, uint32_t playSource ) [inline], [static]`

## Parameters

|                   |                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                                 |
| <i>playSource</i> | audio codec play source, can be a value or combine value of <code>_codec_play_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

**54.7.6.2.20** `static status_t HAL_CODEC_ModuleControl ( void * handle, uint32_t cmd, uint32_t data ) [inline], [static]`

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

## Parameters

|               |                                                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                                                     |
| <i>cmd</i>    | module control cmd, reference <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.

## 54.8 WM8904 Driver

### 54.8.1 Overview

The wm8904 driver provides a codec control interface.

### 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.*
- #define `WM8904_MAP_HEADPHONE_LINEOUT_MAX_VOLUME` 0x3FU  
*WM8904 maximum volume.*

### Enumerations

- enum {  
`kStatus_WM8904_Success` = 0x0,  
`kStatus_WM8904_Fail` = 0x1 }  
*WM8904 status return codes.*
- enum {  
`kWM8904_LRCPolarityNormal` = 0U,  
`kWM8904_LRCPolarityInverted` = 1U << 4U }  
*WM8904 lrc polarity.*
- enum `wm8904_module_t` {

- ```

kWM8904_ModuleADC = 0,
kWM8904_ModuleDAC = 1,
kWM8904_ModulePGA = 2,
kWM8904_ModuleHeadphone = 3,
kWM8904_ModuleLineout = 4 }
    wm8904 module value

```
- enum
 - wm8904 play channel*
 - enum `wm8904_timeslot_t` {


```

kWM8904_TimeSlot0 = 0U,
kWM8904_TimeSlot1 = 1U }
          WM8904 time slot.

```
 - enum `wm8904_protocol_t` {


```

kWM8904_ProtocolI2S = 0x2,
kWM8904_ProtocolLeftJustified = 0x1,
kWM8904_ProtocolRightJustified = 0x0,
kWM8904_ProtocolPCMA = 0x3,
kWM8904_ProtocolPCMB = 0x3 | (1 << 4) }
          The audio data transfer protocol.

```
 - enum `wm8904_fs_ratio_t` {


```

kWM8904_FsRatio64X = 0x0,
kWM8904_FsRatio128X = 0x1,
kWM8904_FsRatio192X = 0x2,
kWM8904_FsRatio256X = 0x3,
kWM8904_FsRatio384X = 0x4,
kWM8904_FsRatio512X = 0x5,
kWM8904_FsRatio768X = 0x6,
kWM8904_FsRatio1024X = 0x7,
kWM8904_FsRatio1408X = 0x8,
kWM8904_FsRatio1536X = 0x9 }
          The SYSCLK / fs ratio.

```
 - enum `wm8904_sample_rate_t` {


```

kWM8904_SampleRate8kHz = 0x0,
kWM8904_SampleRate12kHz = 0x1,
kWM8904_SampleRate16kHz = 0x2,
kWM8904_SampleRate24kHz = 0x3,
kWM8904_SampleRate32kHz = 0x4,
kWM8904_SampleRate48kHz = 0x5,
kWM8904_SampleRate11025Hz = 0x6,
kWM8904_SampleRate22050Hz = 0x7,
kWM8904_SampleRate44100Hz = 0x8 }
          Sample rate.

```
 - enum `wm8904_bit_width_t` {


```

kWM8904_BitWidth16 = 0x0,
kWM8904_BitWidth20 = 0x1,
kWM8904_BitWidth24 = 0x2,
```

- ```
kWM8904_BitWidth32 = 0x3 }
```
- Bit width.*
- enum {
 

```
kWM8904_RecordSourceDifferentialLine = 1U,
kWM8904_RecordSourceLineInput = 2U,
kWM8904_RecordSourceDifferentialMic = 4U,
kWM8904_RecordSourceDigitalMic = 8U }
```

*wm8904 record source*
  - enum {
 

```
kWM8904_RecordChannelLeft1 = 1U,
kWM8904_RecordChannelLeft2 = 2U,
kWM8904_RecordChannelLeft3 = 4U,
kWM8904_RecordChannelRight1 = 1U,
kWM8904_RecordChannelRight2 = 2U,
kWM8904_RecordChannelRight3 = 4U,
kWM8904_RecordChannelDifferentialPositive1 = 1U,
kWM8904_RecordChannelDifferentialPositive2 = 2U,
kWM8904_RecordChannelDifferentialPositive3 = 4U,
kWM8904_RecordChannelDifferentialNegative1 = 8U,
kWM8904_RecordChannelDifferentialNegative2 = 16U,
kWM8904_RecordChannelDifferentialNegative3 = 32U }
```

*wm8904 record channel*
  - enum {
 

```
kWM8904_PlaySourcePGA = 1U,
kWM8904_PlaySourceDAC = 4U }
```

*wm8904 play source*
  - enum `wm8904_sys_clk_source_t` {
 

```
kWM8904_SysClkSourceMCLK = 0U,
kWM8904_SysClkSourceFLL = 1U << 14 }
```

*wm8904 system clock source*
  - enum `wm8904_fll_clk_source_t` { `kWM8904_FLLClkSourceMCLK = 0U` }
 

*wm8904 fll clock source*

## Functions

- `status_t WM8904_WriteRegister (wm8904_handle_t *handle, uint8_t reg, uint16_t value)`  
*WM8904 write register.*
- `status_t WM8904_ReadRegister (wm8904_handle_t *handle, uint8_t reg, uint16_t *value)`  
*WM8904 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 *wm8904Config)`  
*Initializes WM8904.*
- `status_t WM8904_Deinit (wm8904_handle_t *handle)`  
*Deinitializes the WM8904 codec.*
- `void WM8904_GetDefaultConfig (wm8904_config_t *config)`

- Fills the configuration structure with default values.*

  - [status\\_t WM8904\\_SetMasterSlave](#) ([wm8904\\_handle\\_t](#) \*handle, bool master)
    - Sets WM8904 as master or slave.*
  - [status\\_t WM8904\\_SetMasterClock](#) ([wm8904\\_handle\\_t](#) \*handle, [uint32\\_t](#) sysclk, [uint32\\_t](#) sampleRate, [uint32\\_t](#) bitWidth)
    - Sets WM8904 master clock configuration.*
  - [status\\_t WM8904\\_SetFLLConfig](#) ([wm8904\\_handle\\_t](#) \*handle, [wm8904\\_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)
    - SET the module output power.*
  - [status\\_t WM8904\\_SetDACVolume](#) ([wm8904\\_handle\\_t](#) \*handle, [uint8\\_t](#) volume)
    - SET the DAC module volume.*
  - [status\\_t WM8904\\_SetChannelVolume](#) ([wm8904\\_handle\\_t](#) \*handle, [uint32\\_t](#) channel, [uint32\\_t](#) volume)
    - Sets the channel output volume.*
  - [status\\_t WM8904\\_SetRecord](#) ([wm8904\\_handle\\_t](#) \*handle, [uint32\\_t](#) recordSource)
    - SET the WM8904 record source.*
  - [status\\_t WM8904\\_SetRecordChannel](#) ([wm8904\\_handle\\_t](#) \*handle, [uint32\\_t](#) leftRecordChannel, [uint32\\_t](#) rightRecordChannel)
    - SET the WM8904 record source.*
  - [status\\_t WM8904\\_SetPlay](#) ([wm8904\\_handle\\_t](#) \*handle, [uint32\\_t](#) playSource)
    - SET the WM8904 play source.*
  - [status\\_t WM8904\\_SetChannelMute](#) ([wm8904\\_handle\\_t](#) \*handle, [uint32\\_t](#) channel, bool isMute)
    - Sets the channel mute.*

## Driver version

- #define `FSL_WM8904_DRIVER_VERSION` (`MAKE_VERSION(2, 5, 1)`)  
*WM8904 driver version 2.5.1.*

## 54.8.2 Data Structure Documentation

### 54.8.2.1 struct `wm8904_fll_config_t`

#### Data Fields

- `wm8904_fll_clk_source_t` `source`  
*fll reference clock source*
- `uint32_t` `refClock_HZ`  
*fll reference clock frequency*
- `uint32_t` `outputClock_HZ`  
*fll output clock frequency*

### 54.8.2.2 struct `wm8904_audio_format_t`

#### Data Fields

- `wm8904_fs_ratio_t` `fsRatio`  
*SYSCLK / fs ratio.*
- `wm8904_sample_rate_t` `sampleRate`  
*Sample rate.*
- `wm8904_bit_width_t` `bitWidth`  
*Bit width.*

### 54.8.2.3 struct `wm8904_config_t`

#### Data Fields

- `bool` `master`  
*Master or slave.*
- `wm8904_sys_clk_source_t` `sysClkSource`  
*system clock source*
- `wm8904_fll_config_t` \* `fll`  
*fll configuration*
- `wm8904_protocol_t` `protocol`  
*Audio transfer protocol.*
- `wm8904_audio_format_t` `format`  
*Audio format.*
- `uint32_t` `mclk_HZ`  
*MCLK frequency value.*
- `uint16_t` `recordSource`  
*record source*

- uint16\_t `recordChannelLeft`  
*record channel*
- uint16\_t `recordChannelRight`  
*record channel*
- uint16\_t `playSource`  
*play source*
- uint8\_t `slaveAddress`  
*code device slave address*
- `codec_i2c_config_t` `i2cConfig`  
*i2c bus configuration*

#### 54.8.2.4 struct `wm8904_handle_t`

##### Data Fields

- `wm8904_config_t * config`  
*wm8904 config pointer*
- uint8\_t `i2cHandle` [`WM8904_I2C_HANDLER_SIZE`]  
*i2c handle*

#### 54.8.3 Macro Definition Documentation

54.8.3.1 `#define FSL_WM8904_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))`

54.8.3.2 `#define WM8904_I2C_ADDRESS (0x1A)`

54.8.3.3 `#define WM8904_I2C_BITRATE (400000U)`

#### 54.8.4 Enumeration Type Documentation

##### 54.8.4.1 anonymous enum

Enumerator

*kStatus\_WM8904\_Success* Success.  
*kStatus\_WM8904\_Fail* Failure.

##### 54.8.4.2 anonymous enum

Enumerator

*kWM8904\_LRCPolarityNormal* LRC polarity normal.  
*kWM8904\_LRCPolarityInverted* LRC polarity inverted.

### 54.8.4.3 enum wm8904\_module\_t

Enumerator

*kWM8904\_ModuleADC* module ADC  
*kWM8904\_ModuleDAC* module DAC  
*kWM8904\_ModulePGA* module PGA  
*kWM8904\_ModuleHeadphone* module headphone  
*kWM8904\_ModuleLineout* module line out

### 54.8.4.4 anonymous enum

### 54.8.4.5 enum wm8904\_timeslot\_t

Enumerator

*kWM8904\_TimeSlot0* time slot0  
*kWM8904\_TimeSlot1* time slot1

### 54.8.4.6 enum wm8904\_protocol\_t

Enumerator

*kWM8904\_ProtocolI2S* I2S type.  
*kWM8904\_ProtocolLeftJustified* Left justified mode.  
*kWM8904\_ProtocolRightJustified* Right justified mode.  
*kWM8904\_ProtocolPCMA* PCM A mode.  
*kWM8904\_ProtocolPCMB* PCM B mode.

### 54.8.4.7 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.



#### 54.8.4.8 enum wm8904\_sample\_rate\_t

Enumerator

*kWM8904\_SampleRate8kHz* 8 kHz  
*kWM8904\_SampleRate12kHz* 12kHz  
*kWM8904\_SampleRate16kHz* 16kHz  
*kWM8904\_SampleRate24kHz* 24kHz  
*kWM8904\_SampleRate32kHz* 32kHz  
*kWM8904\_SampleRate48kHz* 48kHz  
*kWM8904\_SampleRate11025Hz* 11.025kHz  
*kWM8904\_SampleRate22050Hz* 22.05kHz  
*kWM8904\_SampleRate44100Hz* 44.1kHz

#### 54.8.4.9 enum wm8904\_bit\_width\_t

Enumerator

*kWM8904\_BitWidth16* 16 bits  
*kWM8904\_BitWidth20* 20 bits  
*kWM8904\_BitWidth24* 24 bits  
*kWM8904\_BitWidth32* 32 bits

#### 54.8.4.10 anonymous enum

Enumerator

*kWM8904\_RecordSourceDifferentialLine* record source from differential line  
*kWM8904\_RecordSourceLineInput* record source from line input  
*kWM8904\_RecordSourceDifferentialMic* record source from differential mic  
*kWM8904\_RecordSourceDigitalMic* record source from digital microphone

#### 54.8.4.11 anonymous enum

Enumerator

*kWM8904\_RecordChannelLeft1* left record channel 1  
*kWM8904\_RecordChannelLeft2* left record channel 2  
*kWM8904\_RecordChannelLeft3* left record channel 3  
*kWM8904\_RecordChannelRight1* right record channel 1  
*kWM8904\_RecordChannelRight2* right record channel 2  
*kWM8904\_RecordChannelRight3* right record channel 3  
*kWM8904\_RecordChannelDifferentialPositive1* differential positive record channel 1  
*kWM8904\_RecordChannelDifferentialPositive2* differential positive record channel 2  
*kWM8904\_RecordChannelDifferentialPositive3* differential positive record channel 3

*kWM8904\_RecordChannelDifferentialNegative1* differential negative record channel 1  
*kWM8904\_RecordChannelDifferentialNegative2* differential negative record channel 2  
*kWM8904\_RecordChannelDifferentialNegative3* differential negative record channel 3

#### 54.8.4.12 anonymous enum

Enumerator

*kWM8904\_PlaySourcePGA* play source PGA, bypass ADC  
*kWM8904\_PlaySourceDAC* play source Input3

#### 54.8.4.13 enum wm8904\_sys\_clk\_source\_t

Enumerator

*kWM8904\_SysClkSourceMCLK* wm8904 system clock source from MCLK  
*kWM8904\_SysClkSourceFLL* wm8904 system clock source from FLL

#### 54.8.4.14 enum wm8904\_fl\_clk\_source\_t

Enumerator

*kWM8904\_FLLClkSourceMCLK* wm8904 FLL clock source from MCLK

### 54.8.5 Function Documentation

#### 54.8.5.1 status\_t WM8904\_WriteRegister ( wm8904\_handle\_t \* handle, uint8\_t reg, uint16\_t value )

Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8904 handle structure. |
| <i>reg</i>    | register address.        |
| <i>value</i>  | value to write.          |

Returns

kStatus\_Success, else failed.

#### 54.8.5.2 status\_t WM8904\_ReadRegister ( wm8904\_handle\_t \* handle, uint8\_t reg, uint16\_t \* value )

## Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8904 handle structure. |
| <i>reg</i>    | register address.        |
| <i>value</i>  | value to read.           |

## Returns

kStatus\_Success, else failed.

#### 54.8.5.3 status\_t WM8904\_ModifyRegister ( wm8904\_handle\_t \* *handle*, uint8\_t *reg*, uint16\_t *mask*, uint16\_t *value* )

## Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8904 handle structure. |
| <i>reg</i>    | register address.        |
| <i>mask</i>   | register bits mask.      |
| <i>value</i>  | value to write.          |

## Returns

kStatus\_Success, else failed.

#### 54.8.5.4 status\_t WM8904\_Init ( wm8904\_handle\_t \* *handle*, wm8904\_config\_t \* *wm8904Config* )

## Parameters

|                     |                                 |
|---------------------|---------------------------------|
| <i>handle</i>       | WM8904 handle structure.        |
| <i>wm8904Config</i> | WM8904 configuration structure. |

#### 54.8.5.5 status\_t WM8904\_Deinit ( wm8904\_handle\_t \* *handle* )

This function resets WM8904.

## Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8904 handle structure. |
|---------------|--------------------------|

## Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 54.8.5.6 void WM8904\_GetDefaultConfig ( wm8904\_config\_t \* *config* )

The default values are:

master = false; protocol = kWM8904\_ProtocolI2S; format.fsRatio = kWM8904\_FsRatio64X; format.sampleRate = kWM8904\_SampleRate48kHz; format.bitWidth = kWM8904\_BitWidth16;

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>config</i> | default configurations of wm8904. |
|---------------|-----------------------------------|

#### 54.8.5.7 status\_t WM8904\_SetMasterSlave ( wm8904\_handle\_t \* *handle*, bool *master* )

**Deprecated** DO NOT USE THIS API ANYMORE. IT HAS BEEN SUPERCEDED BY [WM8904\\_SetMasterClock](#)

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>handle</i> | WM8904 handle structure.          |
| <i>master</i> | true for master, false for slave. |

## Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 54.8.5.8 status\_t WM8904\_SetMasterClock ( wm8904\_handle\_t \* *handle*, uint32\_t *sysclk*, uint32\_t *sampleRate*, uint32\_t *bitWidth* )

User should pay attention to the sysclk parameter ,When using external MCLK as system clock source, the value should be frequency of MCLK, when using FLL as system clock source, the value should be frequency of the output of FLL.

## Parameters

|                   |                                |
|-------------------|--------------------------------|
| <i>handle</i>     | WM8904 handle structure.       |
| <i>sysclk</i>     | system clock source frequency. |
| <i>sampleRate</i> | sample rate                    |
| <i>bitWidth</i>   | bit width                      |

## Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 54.8.5.9 status\_t WM8904\_SetFLLConfig ( wm8904\_handle\_t \* *handle*, wm8904\_fl\_config\_t \* *config* )

## Parameters

|               |                            |
|---------------|----------------------------|
| <i>handle</i> | wm8904 handler pointer.    |
| <i>config</i> | FLL configuration pointer. |

#### 54.8.5.10 status\_t WM8904\_SetProtocol ( wm8904\_handle\_t \* *handle*, wm8904\_protocol\_t *protocol* )

## Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>handle</i>   | WM8904 handle structure. |
| <i>protocol</i> | Audio transfer protocol. |

## Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 54.8.5.11 status\_t WM8904\_SetAudioFormat ( wm8904\_handle\_t \* *handle*, uint32\_t *sysclk*, uint32\_t *sampleRate*, uint32\_t *bitWidth* )

User should pay attention to the *sysclk* parameter ,When using external MCLK as system clock source, the value should be frequency of MCLK, when using FLL as system clock source, the value should be frequency of the output of FLL.

## Parameters

|                   |                                |
|-------------------|--------------------------------|
| <i>handle</i>     | WM8904 handle structure.       |
| <i>sysclk</i>     | system clock source frequency. |
| <i>sampleRate</i> | Sample rate frequency in Hz.   |
| <i>bitWidth</i>   | Audio data bit width.          |

## Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 54.8.5.12 **status\_t WM8904\_CheckAudioFormat ( wm8904\_handle\_t \* *handle*, wm8904\_audio\_format\_t \* *format*, uint32\_t *mclkFreq* )**

This api is used check the fsRatio setting based on the mclk and sample rate, if fsRatio setting is not correct, it will correct it according to mclk and sample rate.

## Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>handle</i>   | WM8904 handle structure. |
| <i>format</i>   | audio data format        |
| <i>mclkFreq</i> | mclk frequency           |

## Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 54.8.5.13 **status\_t WM8904\_SetVolume ( wm8904\_handle\_t \* *handle*, uint16\_t *volumeLeft*, uint16\_t *volumeRight* )**

The parameter should be from 0 to 63. The resulting volume will be. 0 for -57DB, 63 for 6DB.

## Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8904 handle structure. |
|---------------|--------------------------|

|                    |                       |
|--------------------|-----------------------|
| <i>volumeLeft</i>  | left channel volume.  |
| <i>volumeRight</i> | right channel volume. |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 54.8.5.14 **status\_t WM8904\_SetMute ( wm8904\_handle\_t \* *handle*, bool *muteLeft*, bool *muteRight* )**

Parameters

|                  |                                              |
|------------------|----------------------------------------------|
| <i>handle</i>    | WM8904 handle structure.                     |
| <i>muteLeft</i>  | true to mute left channel, false to unmute.  |
| <i>muteRight</i> | true to mute right channel, false to unmute. |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 54.8.5.15 **status\_t WM8904\_SelectLRCPolarity ( wm8904\_handle\_t \* *handle*, uint32\_t *polarity* )**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>handle</i>   | WM8904 handle structure. |
| <i>polarity</i> | LRC clock polarity.      |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 54.8.5.16 **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.

#### 54.8.5.17 **status\_t WM8904\_EnableADCTDMMMode ( wm8904\_handle\_t \* *handle*, wm8904\_timeslot\_t *timeSlot* )**

## Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>handle</i>   | WM8904 handle structure. |
| <i>timeSlot</i> | timeslot number.         |

## Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 54.8.5.18 **status\_t WM8904\_SetModulePower ( wm8904\_handle\_t \* *handle*, wm8904\_module\_t *module*, bool *isEnabled* )**

## Parameters

|                       |                                   |
|-----------------------|-----------------------------------|
| <i>handle</i>         | WM8904 handle structure.          |
| <i>module</i>         | wm8904 module.                    |
| <i>isEnabled,true</i> | is power on, false is power down. |

## Returns

kStatus\_WM8904\_Success if successful, different code otherwise..

#### 54.8.5.19 **status\_t WM8904\_SetDACVolume ( wm8904\_handle\_t \* *handle*, uint8\_t *volume* )**



## Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8904 handle structure. |
| <i>volume</i> | volume to be configured. |

## Returns

kStatus\_WM8904\_Success if successful, different code otherwise..

#### 54.8.5.20 **status\_t WM8904\_SetChannelVolume ( wm8904\_handle\_t \* *handle*, uint32\_t *channel*, uint32\_t *volume* )**

The parameter should be from 0 to 63. The resulting volume will be. 0 for -57dB, 63 for 6DB.

## Parameters

|                |                          |
|----------------|--------------------------|
| <i>handle</i>  | codec handle structure.  |
| <i>channel</i> | codec channel.           |
| <i>volume</i>  | volume value from 0 -63. |

## Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 54.8.5.21 **status\_t WM8904\_SetRecord ( wm8904\_handle\_t \* *handle*, uint32\_t *recordSource* )**

## Parameters

|                     |                                                                                                                                                                                                    |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>       | WM8904 handle structure.                                                                                                                                                                           |
| <i>recordSource</i> | record source , can be a value of kCODEC_ModuleRecordSourceDifferentialLine, kCODEC_ModuleRecordSourceDifferentialMic, kCODEC_ModuleRecordSourceSingleEndMic, kCODEC_ModuleRecordSourceDigitalMic. |

## Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 54.8.5.22 **status\_t WM8904\_SetRecordChannel ( wm8904\_handle\_t \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* )**

## Parameters

|                            |                                                                                                                                                                                                            |
|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | WM8904 handle structure.                                                                                                                                                                                   |
| <i>leftRecord-Channel</i>  | channel number of left record channel when using differential source, channel number of single end left channel when using single end source, channel number of digital mic when using digital mic source. |
| <i>rightRecord-Channel</i> | channel number of right record channel when using differential source, channel number of single end right channel when using single end source.                                                            |

## Returns

kStatus\_WM8904\_Success if successful, different code otherwise..

#### 54.8.5.23 status\_t WM8904\_SetPlay ( wm8904\_handle\_t \* *handle*, uint32\_t *playSource* )

## Parameters

|                   |                                                                                                                                                                 |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>     | WM8904 handle structure.                                                                                                                                        |
| <i>playSource</i> | play source , can be a value of kCODEC_ModuleHeadphoneSourcePGA, kCODEC_ModuleHeadphoneSourceDAC, kCODEC_ModuleLineoutSourcePGA, kCODEC_ModuleLineoutSourceDAC. |

## Returns

kStatus\_WM8904\_Success if successful, different code otherwise..

#### 54.8.5.24 status\_t WM8904\_SetChannelMute ( wm8904\_handle\_t \* *handle*, uint32\_t *channel*, bool *isMute* )

## Parameters

|                |                             |
|----------------|-----------------------------|
| <i>handle</i>  | codec handle structure.     |
| <i>channel</i> | codec module name.          |
| <i>isMute</i>  | true is mute, false unmute. |

## Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

## 54.8.6 WM8904 Adapter

### 54.8.6.1 Overview

The wm8904 adapter provides a codec unify control interface.

#### Macros

- #define [HAL\\_CODEC\\_WM8904\\_HANDLER\\_SIZE](#) ([WM8904\\_I2C\\_HANDLER\\_SIZE](#) + 4)  
*codec handler size*

#### Functions

- [status\\_t HAL\\_CODEC\\_WM8904\\_Init](#) (void \*handle, void \*config)  
*Codec initialization.*
- [status\\_t HAL\\_CODEC\\_WM8904\\_Deinit](#) (void \*handle)  
*Codec de-initialization.*
- [status\\_t HAL\\_CODEC\\_WM8904\\_SetFormat](#) (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- [status\\_t HAL\\_CODEC\\_WM8904\\_SetVolume](#) (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- [status\\_t HAL\\_CODEC\\_WM8904\\_SetMute](#) (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- [status\\_t HAL\\_CODEC\\_WM8904\\_SetPower](#) (void \*handle, uint32\_t module, bool powerOn)  
*set audio codec module power.*
- [status\\_t HAL\\_CODEC\\_WM8904\\_SetRecord](#) (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- [status\\_t HAL\\_CODEC\\_WM8904\\_SetRecordChannel](#) (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- [status\\_t HAL\\_CODEC\\_WM8904\\_SetPlay](#) (void \*handle, uint32\_t playSource)  
*codec set play source.*
- [status\\_t HAL\\_CODEC\\_WM8904\\_ModuleControl](#) (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*
- static [status\\_t HAL\\_CODEC\\_Init](#) (void \*handle, void \*config)  
*Codec initialization.*
- static [status\\_t HAL\\_CODEC\\_Deinit](#) (void \*handle)  
*Codec de-initialization.*
- static [status\\_t HAL\\_CODEC\\_SetFormat](#) (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- static [status\\_t HAL\\_CODEC\\_SetVolume](#) (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- static [status\\_t HAL\\_CODEC\\_SetMute](#) (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- static [status\\_t HAL\\_CODEC\\_SetPower](#) (void \*handle, uint32\_t module, bool powerOn)

- *set audio codec module power.*  
static [status\\_t HAL\\_CODEC\\_SetRecord](#) (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- static [status\\_t HAL\\_CODEC\\_SetRecordChannel](#) (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- static [status\\_t HAL\\_CODEC\\_SetPlay](#) (void \*handle, uint32\_t playSource)  
*codec set play source.*
- static [status\\_t HAL\\_CODEC\\_ModuleControl](#) (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

## 54.8.6.2 Function Documentation

### 54.8.6.2.1 status\_t HAL\_CODEC\_WM8904\_Init ( void \* handle, void \* config )

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

Returns

kStatus\_Success is success, else initial failed.

### 54.8.6.2.2 status\_t HAL\_CODEC\_WM8904\_Deinit ( void \* handle )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

### 54.8.6.2.3 status\_t HAL\_CODEC\_WM8904\_SetFormat ( void \* handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth )

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.8.6.2.4 status\_t HAL\_CODEC\_WM8904\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

## Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.8.6.2.5 status\_t HAL\_CODEC\_WM8904\_SetMute ( void \* *handle*, uint32\_t *playChannel*, bool *isMute* )

## Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.8.6.2.6 status\_t HAL\_CODEC\_WM8904\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* )

## Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.8.6.2.7 status\_t HAL\_CODEC\_WM8904\_SetRecord ( void \* *handle*, uint32\_t *recordSource* )

## Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.8.6.2.8 status\_t HAL\_CODEC\_WM8904\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* )

## Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

## Returns

kStatus\_Success is success, else configure failed.

#### 54.8.6.2.9 status\_t HAL\_CODEC\_WM8904\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

## Parameters

|                   |                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                                 |
| <i>playSource</i> | audio codec play source, can be a value or combine value of <code>_codec_play_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

#### 54.8.6.2.10 `status_t HAL_CODEC_WM8904_ModuleControl ( void * handle, uint32_t cmd, uint32_t data )`

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

## Parameters

|               |                                                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                                                     |
| <i>cmd</i>    | module control cmd, reference <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.

#### 54.8.6.2.11 `static status_t HAL_CODEC_Init ( void * handle, void * config ) [inline], [static]`

## Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

## Returns

`kStatus_Success` is success, else initial failed.

#### 54.8.6.2.12 `static status_t HAL_CODEC_Deinit ( void * handle ) [inline], [static]`

## Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

## Returns

kStatus\_Success is success, else de-initial failed.

**54.8.6.2.13** `static status_t HAL_CODEC_SetFormat ( void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth ) [inline], [static]`

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

## Returns

kStatus\_Success is success, else configure failed.

**54.8.6.2.14** `static status_t HAL_CODEC_SetVolume ( void * handle, uint32_t playChannel, uint32_t volume ) [inline], [static]`

## Parameters

|                    |                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                                   |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of <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.

**54.8.6.2.15** `static status_t HAL_CODEC_SetMute ( void * handle, uint32_t playChannel, bool isMute ) [inline], [static]`



## Parameters

|                    |                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                                   |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> . |
| <i>isMute</i>      | true is mute, false is unmute.                                                                  |

## Returns

`kStatus_Success` is success, else configure failed.

**54.8.6.2.16** `static status_t HAL_CODEC_SetPower ( void * handle, uint32_t module, bool powerOn ) [inline], [static]`

## Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

## Returns

`kStatus_Success` is success, else configure failed.

**54.8.6.2.17** `static status_t HAL_CODEC_SetRecord ( void * handle, uint32_t recordSource ) [inline], [static]`

## Parameters

|                     |                                                                                                   |
|---------------------|---------------------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                                     |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of <code>_codec_record_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

**54.8.6.2.18** `static status_t HAL_CODEC_SetRecordChannel ( void * handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel ) [inline], [static]`

## Parameters

|                            |                                                                                                                                                              |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                                                |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference <code>_codec_record_channel</code> , can be a value or combine value of member in <code>_codec_record_channel</code> . |
| <i>rightRecord-Channel</i> | audio codec record channel, reference <code>_codec_record_channel</code> , can be a value or combine value of member in <code>_codec_record_channel</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

**54.8.6.2.19** `static status_t HAL_CODEC_SetPlay ( void * handle, uint32_t playSource ) [inline], [static]`

## Parameters

|                   |                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                                 |
| <i>playSource</i> | audio codec play source, can be a value or combine value of <code>_codec_play_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

**54.8.6.2.20** `static status_t HAL_CODEC_ModuleControl ( void * handle, uint32_t cmd, uint32_t data ) [inline], [static]`

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

## Parameters

|               |                                                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                                                     |
| <i>cmd</i>    | module control cmd, reference <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 55

## Serial Manager

### 55.1 Overview

This chapter describes the programming interface of the serial manager component.

The serial manager component provides a series of APIs to operate different serial port types. The port types it supports are UART, USB CDC and SWO.

### Modules

- [Serial Port SWO](#)
- [Serial Port USB](#)
- [Serial Port Uart](#)

### Data Structures

- struct [serial\\_manager\\_config\\_t](#)  
*serial manager config structure [More...](#)*
- struct [serial\\_manager\\_callback\\_message\\_t](#)  
*Callback message structure. [More...](#)*

### Macros

- #define [SERIAL\\_MANAGER\\_NON\\_BLOCKING\\_MODE](#) (1U)  
*Enable or disable serial manager non-blocking mode (1 - enable, 0 - disable)*
- #define [SERIAL\\_MANAGER\\_RING\\_BUFFER\\_FLOWCONTROL](#) (0U)  
*Enable or ring buffer flow control (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_UART](#) (0U)  
*Enable or disable uart port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_UART\\_DMA](#) (0U)  
*Enable or disable uart dma port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_USBCDC](#) (0U)  
*Enable or disable USB CDC port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_SWO](#) (0U)  
*Enable or disable SWO port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_VIRTUAL](#) (0U)  
*Enable or disable USB CDC virtual port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_RPMSG](#) (0U)  
*Enable or disable rPMSG port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_SPI\\_MASTER](#) (0U)  
*Enable or disable SPI Master port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_SPI\\_SLAVE](#) (0U)  
*Enable or disable SPI Slave port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_BLE\\_WU](#) (0U)  
*Enable or disable BLE WU port (1 - enable, 0 - disable)*

- #define `SERIAL_MANAGER_TASK_HANDLE_TX` (0U)  
*Enable or disable SerialManager\_Task() handle TX to prevent recursive calling.*
- #define `SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE` (1U)  
*Set the default delay time in ms used by SerialManager\_WriteTimeDelay().*
- #define `SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE` (1U)  
*Set the default delay time in ms used by SerialManager\_ReadTimeDelay().*
- #define `SERIAL_MANAGER_TASK_HANDLE_RX_AVAILABLE_NOTIFY` (0U)  
*Enable or disable SerialManager\_Task() handle RX data available notify.*
- #define `SERIAL_MANAGER_WRITE_HANDLE_SIZE` (44U)  
*Set serial manager write handle size.*
- #define `SERIAL_MANAGER_USE_COMMON_TASK` (0U)  
*SERIAL\_PORT\_UART\_HANDLE\_SIZE/SERIAL\_PORT\_USB\_CDC\_HANDLE\_SIZE + serial manager dedicated size.*
- #define `SERIAL_MANAGER_HANDLE_SIZE` (SERIAL\_MANAGER\_HANDLE\_SIZE\_TEMP + 124U)  
*Definition of serial manager handle size.*
- #define `SERIAL_MANAGER_HANDLE_DEFINE`(name) uint32\_t name[(((SERIAL\_MANAGER\_HANDLE\_SIZE + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t)))]  
*Defines the serial manager handle.*
- #define `SERIAL_MANAGER_WRITE_HANDLE_DEFINE`(name) uint32\_t name[(((SERIAL\_MANAGER\_WRITE\_HANDLE\_SIZE + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t)))]  
*Defines the serial manager write handle.*
- #define `SERIAL_MANAGER_READ_HANDLE_DEFINE`(name) uint32\_t name[(((SERIAL\_MANAGER\_READ\_HANDLE\_SIZE + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t)))]  
*Defines the serial manager read handle.*
- #define `SERIAL_MANAGER_TASK_PRIORITY` (2U)  
*Macro to set serial manager task priority.*
- #define `SERIAL_MANAGER_TASK_STACK_SIZE` (1000U)  
*Macro to set serial manager task stack size.*

## Typedefs

- typedef void \* `serial_handle_t`  
*The handle of the serial manager module.*
- typedef void \* `serial_write_handle_t`  
*The write handle of the serial manager module.*
- typedef void \* `serial_read_handle_t`  
*The read handle of the serial manager module.*
- typedef void(\* `serial_manager_callback_t` )(void \*callbackParam, `serial_manager_callback_message_t` \*message, `serial_manager_status_t` status)  
*serial manager callback function*
- typedef int32\_t(\* `serial_manager_lowpower_critical_callback_t` )(int32\_t power\_mode)  
*serial manager Lowpower Critical callback function*

## Enumerations

- enum `serial_port_type_t` {  
`kSerialPort_None` = 0U,  
`kSerialPort_Uart` = 1U,  
`kSerialPort_UsbCdc`,  
`kSerialPort_Swo`,  
`kSerialPort_Virtual`,  
`kSerialPort_Rpmsg`,  
`kSerialPort_UartDma`,  
`kSerialPort_SpiMaster`,  
`kSerialPort_SpiSlave`,  
`kSerialPort_BleWu` }  
*serial port type*
- enum `serial_manager_type_t` {  
`kSerialManager_NonBlocking` = 0x0U,  
`kSerialManager_Blocking` = 0x8F41U }  
*serial manager type*
- enum `serial_manager_status_t` {  
`kStatus_SerialManager_Success` = `kStatus_Success`,  
`kStatus_SerialManager_Error` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 1)`,  
`kStatus_SerialManager_Busy` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 2)`,  
`kStatus_SerialManager_Notify` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 3)`,  
`kStatus_SerialManager_Canceled`,  
`kStatus_SerialManager_HandleConflict` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 5)`,  
`kStatus_SerialManager_RingBufferOverflow`,  
`kStatus_SerialManager_NotConnected` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 7)` }  
*serial manager error code*

## Functions

- `serial_manager_status_t SerialManager_Init` (`serial_handle_t` serialHandle, const `serial_manager_config_t` \*config)  
*Initializes a serial manager module with the serial manager handle and the user configuration structure.*
- `serial_manager_status_t SerialManager_Deinit` (`serial_handle_t` serialHandle)  
*De-initializes the serial manager module instance.*
- `serial_manager_status_t SerialManager_OpenWriteHandle` (`serial_handle_t` serialHandle, `serial_write_handle_t` writeHandle)  
*Opens a writing handle for the serial manager module.*
- `serial_manager_status_t SerialManager_CloseWriteHandle` (`serial_write_handle_t` writeHandle)  
*Closes a writing handle for the serial manager module.*
- `serial_manager_status_t SerialManager_OpenReadHandle` (`serial_handle_t` serialHandle, `serial_read_handle_t` readHandle)  
*Opens a reading handle for the serial manager module.*
- `serial_manager_status_t SerialManager_CloseReadHandle` (`serial_read_handle_t` readHandle)  
*Closes a reading for the serial manager module.*

- `serial_manager_status_t SerialManager_WriteBlocking` (`serial_write_handle_t writeHandle`, `uint8_t *buffer`, `uint32_t length`)  
*Transmits data with the blocking mode.*
- `serial_manager_status_t SerialManager_ReadBlocking` (`serial_read_handle_t readHandle`, `uint8_t *buffer`, `uint32_t length`)  
*Reads data with the blocking mode.*
- `serial_manager_status_t SerialManager_WriteNonBlocking` (`serial_write_handle_t writeHandle`, `uint8_t *buffer`, `uint32_t length`)  
*Transmits data with the non-blocking mode.*
- `serial_manager_status_t SerialManager_ReadNonBlocking` (`serial_read_handle_t readHandle`, `uint8_t *buffer`, `uint32_t length`)  
*Reads data with the non-blocking mode.*
- `serial_manager_status_t SerialManager_TryRead` (`serial_read_handle_t readHandle`, `uint8_t *buffer`, `uint32_t length`, `uint32_t *receivedLength`)  
*Tries to read data.*
- `serial_manager_status_t SerialManager_CancelWriting` (`serial_write_handle_t writeHandle`)  
*Cancels unfinished send transmission.*
- `serial_manager_status_t SerialManager_CancelReading` (`serial_read_handle_t readHandle`)  
*Cancels unfinished receive transmission.*
- `serial_manager_status_t SerialManager_InstallTxCallback` (`serial_write_handle_t writeHandle`, `serial_manager_callback_t callback`, `void *callbackParam`)  
*Installs a TX callback and callback parameter.*
- `serial_manager_status_t SerialManager_InstallRxCallback` (`serial_read_handle_t readHandle`, `serial_manager_callback_t callback`, `void *callbackParam`)  
*Installs a RX callback and callback parameter.*
- `static bool SerialManager_needPollingIsr` (`void`)  
*Check if need polling ISR.*
- `serial_manager_status_t SerialManager_EnterLowpower` (`serial_handle_t serialHandle`)  
*Prepares to enter low power consumption.*
- `serial_manager_status_t SerialManager_ExitLowpower` (`serial_handle_t serialHandle`)  
*Restores from low power consumption.*
- `void SerialManager_SetLowpowerCriticalCb` (`const serial_manager_lowpower_critical_CBs_t *pfCallback`)  
*This function performs initialization of the callbacks structure used to disable lowpower when serial manager is active.*

## 55.2 Data Structure Documentation

### 55.2.1 struct serial\_manager\_config\_t

#### Data Fields

- `uint8_t * ringBuffer`  
*Ring buffer address, it is used to buffer data received by the hardware.*
- `uint32_t ringBufferSize`  
*The size of the ring buffer.*
- `serial_port_type_t type`  
*Serial port type.*
- `serial_manager_type_t blockType`

- *Serial manager port type.*  
void \* `portConfig`  
*Serial port configuration.*

### Field Documentation

#### (1) `uint8_t* serial_manager_config_t::ringBuffer`

Besides, the memory space cannot be free during the lifetime of the serial manager module.

## 55.2.2 `struct serial_manager_callback_message_t`

### Data Fields

- `uint8_t * buffer`  
*Transferred buffer.*
- `uint32_t length`  
*Transferred data length.*

## 55.3 Macro Definition Documentation

### 55.3.1 `#define SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE (1U)`

### 55.3.2 `#define SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE (1U)`

### 55.3.3 `#define SERIAL_MANAGER_USE_COMMON_TASK (0U)`

Macro to determine whether use common task.

### 55.3.4 `#define SERIAL_MANAGER_HANDLE_SIZE (SERIAL_MANAGER_HANDLE_SIZE_TEMP + 124U)`

### 55.3.5 `#define SERIAL_MANAGER_HANDLE_DEFINE( name ) uint32_t name[(((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t)))]`

This macro is used to define a 4 byte aligned serial manager handle. Then use "(serial\_handle\_t)name" to get the serial manager handle.

The macro should be global and could be optional. You could also define serial manager handle by yourself.

This is an example,

```
* SERIAL_MANAGER_HANDLE_DEFINE(serialManagerHandle);
*
```

Parameters

|             |                                               |
|-------------|-----------------------------------------------|
| <i>name</i> | The name string of the serial manager handle. |
|-------------|-----------------------------------------------|

### 55.3.6 #define SERIAL\_MANAGER\_WRITE\_HANDLE\_DEFINE( *name* ) uint32\_t name[(((SERIAL\_MANAGER\_WRITE\_HANDLE\_SIZE + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t)))]

This macro is used to define a 4 byte aligned serial manager write handle. Then use "(serial\_write\_handle-*t*)name" to get the serial manager write handle.

The macro should be global and could be optional. You could also define serial manager write handle by yourself.

This is an example,

```
* SERIAL_MANAGER_WRITE_HANDLE_DEFINE(serialManagerwriteHandle);
*
```

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>name</i> | The name string of the serial manager write handle. |
|-------------|-----------------------------------------------------|

### 55.3.7 #define SERIAL\_MANAGER\_READ\_HANDLE\_DEFINE( *name* ) uint32\_t name[(((SERIAL\_MANAGER\_READ\_HANDLE\_SIZE + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t)))]

This macro is used to define a 4 byte aligned serial manager read handle. Then use "(serial\_read\_handle-*t*)name" to get the serial manager read handle.

The macro should be global and could be optional. You could also define serial manager read handle by yourself.

This is an example,

```
* SERIAL_MANAGER_READ_HANDLE_DEFINE(serialManagerReadHandle);
*
```



Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| <i>name</i> | The name string of the serial manager read handle. |
|-------------|----------------------------------------------------|

**55.3.8 #define SERIAL\_MANAGER\_TASK\_PRIORITY (2U)**

**55.3.9 #define SERIAL\_MANAGER\_TASK\_STACK\_SIZE (1000U)**

## 55.4 Enumeration Type Documentation

**55.4.1 enum serial\_port\_type\_t**

Enumerator

*kSerialPort\_None* Serial port is none.  
*kSerialPort\_Uart* Serial port UART.  
*kSerialPort\_UsbCdc* Serial port USB CDC.  
*kSerialPort\_Swo* Serial port SWO.  
*kSerialPort\_Virtual* Serial port Virtual.  
*kSerialPort\_Rpmsg* Serial port RPMSG.  
*kSerialPort\_UartDma* Serial port UART DMA.  
*kSerialPort\_SpiMaster* Serial port SPIMASTER.  
*kSerialPort\_SpiSlave* Serial port SPISLAVE.  
*kSerialPort\_BleWu* Serial port BLE WU.

**55.4.2 enum serial\_manager\_type\_t**

Enumerator

*kSerialManager\_NonBlocking* None blocking handle.  
*kSerialManager\_Blocking* Blocking handle.

**55.4.3 enum serial\_manager\_status\_t**

Enumerator

*kStatus\_SerialManager\_Success* Success.  
*kStatus\_SerialManager\_Error* Failed.  
*kStatus\_SerialManager\_Busy* Busy.  
*kStatus\_SerialManager\_Notify* Ring buffer is not empty.  
*kStatus\_SerialManager\_Canceled* the non-blocking request is canceled

*kStatus\_SerialManager\_HandleConflict* The handle is opened.

*kStatus\_SerialManager\_RingBufferOverflow* The ring buffer is overflowed.

*kStatus\_SerialManager\_NotConnected* The host is not connected.

## 55.5 Function Documentation

### 55.5.1 serial\_manager\_status\_t SerialManager\_Init ( serial\_handle\_t serialHandle, const serial\_manager\_config\_t \* config )

This function configures the Serial Manager module with user-defined settings. The user can configure the configuration structure. The parameter serialHandle is a pointer to point to a memory space of size [SERIAL\\_MANAGER\\_HANDLE\\_SIZE](#) allocated by the caller. The Serial Manager module supports three types of serial port, UART (includes UART, USART, LPSCI, LPUART, etc), USB CDC and swo. Please refer to [serial\\_port\\_type\\_t](#) for serial port setting. These three types can be set by using [serial\\_manager\\_config\\_t](#).

Example below shows how to use this API to configure the Serial Manager. For UART,

```
* #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
* static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
* static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
* serial_manager_config_t config;
* serial_port_uart_config_t uartConfig;
* config.type = kSerialPort_Uart;
* config.ringBuffer = &s_ringBuffer[0];
* config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
* uartConfig.instance = 0;
* uartConfig.clockRate = 24000000;
* uartConfig.baudRate = 115200;
* uartConfig.parityMode = kSerialManager_UartParityDisabled;
* uartConfig.stopBitCount = kSerialManager_UartOneStopBit;
* uartConfig.enableRx = 1;
* uartConfig.enableTx = 1;
* uartConfig.enableRxRTS = 0;
* uartConfig.enableTxCTS = 0;
* config.portConfig = &uartConfig;
* SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*
```

For USB CDC,

```
* #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
* static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
* static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
* serial_manager_config_t config;
* serial_port_usb_cdc_config_t usbCdcConfig;
* config.type = kSerialPort_UsbCdc;
* config.ringBuffer = &s_ringBuffer[0];
* config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
* usbCdcConfig.controllerIndex =
* kSerialManager_UsbControllerKhci0;
* config.portConfig = &usbCdcConfig;
* SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*
```

## Parameters

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>serialHandle</i> | Pointer to point to a memory space of size <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. |

### 55.5.2 serial\_manager\_status\_t SerialManager\_Deinit ( serial\_handle\_t serialHandle )

This function de-initializes the serial manager module instance. If the opened writing or reading handle is not closed, the function will return `kStatus_SerialManager_Busy`.

## Parameters

|                     |                                           |
|---------------------|-------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. |
|---------------------|-------------------------------------------|

## Return values

|                                      |                                                 |
|--------------------------------------|-------------------------------------------------|
| <i>kStatus_SerialManager_Success</i> | The serial manager de-initialization succeed.   |
| <i>kStatus_SerialManager_Busy</i>    | Opened reading or writing handle is not closed. |

### 55.5.3 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);
*
```

**55.5.4 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_</i><br><i>Success</i> | The writing handle is closed. |
|-------------------------------------------------|-------------------------------|

### 55.5.5 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: <a href="#">SERIAL_MANAGER_READ_HANDLE_DEFINE(readHandle)</a> ; or <code>uint32_t readHandle[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code> |

## Return values

|                                                 |                                        |
|-------------------------------------------------|----------------------------------------|
| <i>kStatus_SerialManager_</i><br><i>Error</i>   | An error occurred.                     |
| <i>kStatus_SerialManager_</i><br><i>Success</i> | The reading handle is opened.          |
| <i>kStatus_SerialManager_</i><br><i>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));
*

```

### 55.5.6 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. |
|--------------------------------------|-------------------------------|

### 55.5.7 serial\_manager\_status\_t SerialManager\_WriteBlocking ( serial\_write\_handle\_t writeHandle, uint8\_t \* buffer, uint32\_t length )

This is a blocking function, which polls the sending queue, waits for the sending queue to be empty. This function sends data using an interrupt method. The interrupt of the hardware could not be disabled. And There can only one buffer for transmission for the writing handle at the same time.

Note

The function [SerialManager\\_WriteBlocking](#) and the function `SerialManager_WriteNonBlocking` cannot be used at the same time. And, the function `SerialManager_CancelWriting` cannot be used to abort the transmission of this function.

Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>writeHandle</i> | The serial manager module handle pointer. |
|--------------------|-------------------------------------------|

|               |                                     |
|---------------|-------------------------------------|
| <i>buffer</i> | Start address of the data to write. |
| <i>length</i> | Length of the data to write.        |

Return values

|                                         |                                                                  |
|-----------------------------------------|------------------------------------------------------------------|
| <i>kStatus_SerialManager_ - Success</i> | Successfully sent all data.                                      |
| <i>kStatus_SerialManager_ - Busy</i>    | Previous transmission still not finished; data not all sent yet. |
| <i>kStatus_SerialManager_ - Error</i>   | An error occurred.                                               |

### 55.5.8 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

|                   |                                                       |
|-------------------|-------------------------------------------------------|
| <i>readHandle</i> | The serial manager module handle pointer.             |
| <i>buffer</i>     | Start address of the data to store the received data. |
| <i>length</i>     | The length of the data to be received.                |

Return values

|                                         |                                 |
|-----------------------------------------|---------------------------------|
| <i>kStatus_SerialManager_ - Success</i> | Successfully received all data. |
|-----------------------------------------|---------------------------------|



|                                     |                                                                      |
|-------------------------------------|----------------------------------------------------------------------|
| <i>kStatus_SerialManager_-Busy</i>  | Previous transmission still not finished; data not all received yet. |
| <i>kStatus_SerialManager_-Error</i> | An error occurred.                                                   |

### 55.5.9 serial\_manager\_status\_t SerialManager\_WriteNonBlocking ( serial\_write\_handle\_t writeHandle, uint8\_t \* buffer, uint32\_t length )

This is a non-blocking function, which returns directly without waiting for all data to be sent. When all data is sent, the module notifies the upper layer through a TX callback function and passes the status parameter [kStatus\\_SerialManager\\_Success](#). 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 TX callback is mandatory before the function could be used.

Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>writeHandle</i> | The serial manager module handle pointer. |
| <i>buffer</i>      | Start address of the data to write.       |
| <i>length</i>      | Length of the data to write.              |

Return values

|                                       |                                                                  |
|---------------------------------------|------------------------------------------------------------------|
| <i>kStatus_SerialManager_-Success</i> | Successfully sent all data.                                      |
| <i>kStatus_SerialManager_-Busy</i>    | Previous transmission still not finished; data not all sent yet. |
| <i>kStatus_SerialManager_-Error</i>   | An error occurred.                                               |

### 55.5.10 serial\_manager\_status\_t SerialManager\_ReadNonBlocking ( serial\_read\_handle\_t readHandle, uint8\_t \* buffer, uint32\_t length )

This is a non-blocking function, which returns directly without waiting for all data to be received. When all data is received, the module driver notifies the upper layer through a RX callback function and passes the

status parameter [kStatus\\_SerialManager\\_Success](#). 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 RX callback is mandatory before the function could be used.

Parameters

|                   |                                                       |
|-------------------|-------------------------------------------------------|
| <i>readHandle</i> | The serial manager module handle pointer.             |
| <i>buffer</i>     | Start address of the data to store the received data. |
| <i>length</i>     | The length of the data to be received.                |

Return values

|                                         |                                                                      |
|-----------------------------------------|----------------------------------------------------------------------|
| <i>kStatus_SerialManager_ - Success</i> | Successfully received all data.                                      |
| <i>kStatus_SerialManager_ - Busy</i>    | Previous transmission still not finished; data not all received yet. |
| <i>kStatus_SerialManager_ - Error</i>   | An error occurred.                                                   |

**55.5.11 serial\_manager\_status\_t SerialManager\_TryRead ( serial\_read\_handle\_t readHandle, uint8\_t \* buffer, uint32\_t length, uint32\_t \* receivedLength )**

The function tries to read data from internal ring buffer. If the ring buffer is not empty, the data will be copied from ring buffer to up layer buffer. The copied length is the minimum of the ring buffer and up layer length. After the data is copied, the actual data length is passed by the parameter length. And There can only one buffer for receiving for the reading handle at the same time.

Parameters

|                       |                                                       |
|-----------------------|-------------------------------------------------------|
| <i>readHandle</i>     | The serial manager module handle pointer.             |
| <i>buffer</i>         | Start address of the data to store the received data. |
| <i>length</i>         | The length of the data to be received.                |
| <i>receivedLength</i> | Length received from the ring buffer directly.        |

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.                                                   |

### 55.5.12 serial\_manager\_status\_t SerialManager\_CancelWriting ( serial\_write\_handle\_t writeHandle )

The function cancels unfinished send transmission. When the transfer is canceled, the module notifies the upper layer through a TX callback function and passes the status parameter [kStatus\\_SerialManager\\_Canceled](#).

Note

The function [SerialManager\\_CancelWriting](#) cannot be used to abort the transmission of the function [SerialManager\\_WriteBlocking](#).

Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>writeHandle</i> | The serial manager module handle pointer. |
|--------------------|-------------------------------------------|

Return values

|                                            |                                     |
|--------------------------------------------|-------------------------------------|
| <i>kStatus_SerialManager_-<br/>Success</i> | Get successfully abort the sending. |
| <i>kStatus_SerialManager_-<br/>Error</i>   | An error occurred.                  |

### 55.5.13 serial\_manager\_status\_t SerialManager\_CancelReading ( serial\_read\_handle\_t readHandle )

The function cancels unfinished receive transmission. When the transfer is canceled, the module notifies the upper layer through a RX callback function and passes the status parameter [kStatus\\_SerialManager\\_Canceled](#).

Note

The function [SerialManager\\_CancelReading](#) cannot be used to abort the transmission of the function [SerialManager\\_ReadBlocking](#).

Parameters

|                   |                                           |
|-------------------|-------------------------------------------|
| <i>readHandle</i> | The serial manager module handle pointer. |
|-------------------|-------------------------------------------|

Return values

|                                            |                                       |
|--------------------------------------------|---------------------------------------|
| <i>kStatus_SerialManager_-<br/>Success</i> | Get successfully abort the receiving. |
| <i>kStatus_SerialManager_-<br/>Error</i>   | An error occurred.                    |

#### 55.5.14 **serial\_manager\_status\_t SerialManager\_InstallTxCallback ( serial\_write\_handle\_t writeHandle, serial\_manager\_callback\_t callback, void \* callbackParam )**

This function is used to install the TX callback and callback parameter for the serial manager module. When any status of TX transmission changed, the driver will notify the upper layer by the installed callback function. And the status is also passed as status parameter when the callback is called.

Parameters

|                      |                                           |
|----------------------|-------------------------------------------|
| <i>writeHandle</i>   | The serial manager module handle pointer. |
| <i>callback</i>      | The callback function.                    |
| <i>callbackParam</i> | The parameter of the callback function.   |

Return values

|                                            |                                    |
|--------------------------------------------|------------------------------------|
| <i>kStatus_SerialManager_-<br/>Success</i> | Successfully install the callback. |
|--------------------------------------------|------------------------------------|

#### 55.5.15 **serial\_manager\_status\_t SerialManager\_InstallRxCallback ( serial\_read\_handle\_t readHandle, serial\_manager\_callback\_t callback, void \* callbackParam )**

This function is used to install the RX callback and callback parameter for the serial manager module. When any status of RX transmission changed, the driver will notify the upper layer by the installed callback

function. And the status is also passed as status parameter when the callback is called.

Parameters

|                      |                                           |
|----------------------|-------------------------------------------|
| <i>readHandle</i>    | The serial manager module handle pointer. |
| <i>callback</i>      | The callback function.                    |
| <i>callbackParam</i> | The parameter of the callback function.   |

Return values

|                                            |                                    |
|--------------------------------------------|------------------------------------|
| <i>kStatus_SerialManager_-<br/>Success</i> | Successfully install the callback. |
|--------------------------------------------|------------------------------------|

**55.5.16 static bool SerialManager\_needPollingIsr ( void ) [inline], [static]**

This function is used to check if need polling ISR.

Return values

|             |                  |
|-------------|------------------|
| <i>TRUE</i> | if need polling. |
|-------------|------------------|

**55.5.17 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. |
|--------------------------------------------|-----------------------|

**55.5.18 serial\_manager\_status\_t SerialManager\_ExitLowpower ( serial\_handle\_t serialHandle )**

This function is used to restore from low power consumption.

Parameters

|                     |                                           |
|---------------------|-------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. |
|---------------------|-------------------------------------------|

Return values

|                                      |                       |
|--------------------------------------|-----------------------|
| <i>kStatus_SerialManager_Success</i> | Successful operation. |
|--------------------------------------|-----------------------|

**55.5.19 void SerialManager\_SetLowpowerCriticalCb ( const serial\_manager\_lowpower\_critical\_CBs\_t \* *pfCallback* )**

Parameters

|                   |                                                                   |
|-------------------|-------------------------------------------------------------------|
| <i>pfCallback</i> | Pointer to the function structure used to allow/disable lowpower. |
|-------------------|-------------------------------------------------------------------|

## 55.6 Serial Port Uart

### 55.6.1 Overview

#### Macros

- #define `SERIAL_PORT_UART_DMA_RECEIVE_DATA_LENGTH` (64U)  
*serial port uart handle size*
- #define `SERIAL_USE_CONFIGURE_STRUCTURE` (0U)  
*Enable or disable the configure structure pointer.*

#### Enumerations

- enum `serial_port_uart_parity_mode_t` {  
  `kSerialManager_UartParityDisabled` = 0x0U,  
  `kSerialManager_UartParityEven` = 0x2U,  
  `kSerialManager_UartParityOdd` = 0x3U }  
*serial port uart parity mode*
- enum `serial_port_uart_stop_bit_count_t` {  
  `kSerialManager_UartOneStopBit` = 0U,  
  `kSerialManager_UartTwoStopBit` = 1U }  
*serial port uart stop bit count*

### 55.6.2 Enumeration Type Documentation

#### 55.6.2.1 enum serial\_port\_uart\_parity\_mode\_t

Enumerator

*kSerialManager\_UartParityDisabled* Parity disabled.  
*kSerialManager\_UartParityEven* Parity even enabled.  
*kSerialManager\_UartParityOdd* Parity odd enabled.

#### 55.6.2.2 enum serial\_port\_uart\_stop\_bit\_count\_t

Enumerator

*kSerialManager\_UartOneStopBit* One stop bit.  
*kSerialManager\_UartTwoStopBit* Two stop bits.



## 55.7 Serial Port USB

### 55.7.1 Overview

#### Modules

- [USB Device Configuration](#)

#### Data Structures

- struct [serial\\_port\\_usb\\_cdc\\_config\\_t](#)  
*serial port usb config struct [More...](#)*

#### Macros

- #define [SERIAL\\_PORT\\_USB\\_CDC\\_HANDLE\\_SIZE](#) (72U)  
*serial port usb handle size*
- #define [USB\\_DEVICE\\_INTERRUPT\\_PRIORITY](#) (3U)  
*USB interrupt priority.*

#### Enumerations

- enum [serial\\_port\\_usb\\_cdc\\_controller\\_index\\_t](#) {  
[kSerialManager\\_UsbControllerKhci0](#) = 0U,  
[kSerialManager\\_UsbControllerKhci1](#) = 1U,  
[kSerialManager\\_UsbControllerEhci0](#) = 2U,  
[kSerialManager\\_UsbControllerEhci1](#) = 3U,  
[kSerialManager\\_UsbControllerLpcIp3511Fs0](#) = 4U,  
[kSerialManager\\_UsbControllerLpcIp3511Fs1](#) = 5U,  
[kSerialManager\\_UsbControllerLpcIp3511Hs0](#) = 6U,  
[kSerialManager\\_UsbControllerLpcIp3511Hs1](#) = 7U,  
[kSerialManager\\_UsbControllerOhci0](#) = 8U,  
[kSerialManager\\_UsbControllerOhci1](#) = 9U,  
[kSerialManager\\_UsbControllerIp3516Hs0](#) = 10U,  
[kSerialManager\\_UsbControllerIp3516Hs1](#) = 11U }  
*USB controller ID.*

## 55.7.2 Data Structure Documentation

### 55.7.2.1 struct serial\_port\_usb\_cdc\_config\_t

#### Data Fields

- [serial\\_port\\_usb\\_cdc\\_controller\\_index\\_t controllerIndex](#)  
*controller index*

## 55.7.3 Enumeration Type Documentation

### 55.7.3.1 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.

## **55.7.4 USB Device Configuration**

## 55.8 Serial Port SWO

### 55.8.1 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*

### 55.8.2 Data Structure Documentation

#### 55.8.2.1 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.*

### 55.8.3 Enumeration Type Documentation

#### 55.8.3.1 enum serial\_port\_swo\_protocol\_t

Enumerator

*[kSerialManager\\_SwoProtocolManchester](#)* SWO Manchester protocol.  
*[kSerialManager\\_SwoProtocolNrz](#)* SWO UART/NRZ protocol.

# Chapter 56

## Nic301

### 56.1 Overview

#### Functions

- static void `NIC_SetReadQos` (nic\_reg\_t base, nic\_qos\_t value)  
*Set read\_qos Value.*
- static nic\_qos\_t `NIC_GetReadQos` (nic\_reg\_t base)  
*Get read\_qos Value.*
- static void `NIC_SetWriteQos` (nic\_reg\_t base, nic\_qos\_t value)  
*Set write\_qos Value.*
- static nic\_qos\_t `NIC_GetWriteQos` (nic\_reg\_t base)  
*Get write\_qos Value.*
- static void `NIC_SetFnModAhb` (nic\_reg\_t base, nic\_fn\_mod\_ahb\_t v)  
*Set fn\_mod\_ahb Value.*
- static nic\_fn\_mod\_ahb\_t `NIC_GetFnModAhb` (nic\_reg\_t base)  
*Get fn\_mod\_ahb Value.*
- static void `NIC_SetWrTideMark` (nic\_reg\_t base, uint8\_t value)  
*Set wr\_tidemark Value.*
- static uint8\_t `NIC_GetWrTideMark` (nic\_reg\_t base)  
*Get wr\_tidemark Value.*
- static void `NIC_SetFnMod` (nic\_reg\_t base, nic\_fn\_mod\_t value)  
*Set fn\_mod Value.*
- static nic\_fn\_mod\_t `NIC_GetFnMod` (nic\_reg\_t base)  
*Get fn\_mod Value.*
- static void `NIC_SetFnMod2` (nic\_reg\_t base, nic\_fn\_mod\_t value)  
*Set fn\_mod2 Value.*
- static nic\_fn\_mod2\_t `NIC_GetFnMod2` (nic\_reg\_t base)  
*Get fn\_mod2 Value.*

#### Driver version

- #define `FSL_NIC301_DRIVER_VERSION` (`MAKE_VERSION`(2U, 0U, 1U))  
*NIC301 driver version 2.0.1.*

### 56.2 Macro Definition Documentation

#### 56.2.1 #define FSL\_NIC301\_DRIVER\_VERSION (MAKE\_VERSION(2U, 0U, 1U))

### 56.3 Function Documentation

#### 56.3.1 static void NIC\_SetReadQos ( nic\_reg\_t base, nic\_qos\_t value ) [inline], [static]

Parameters

|              |                             |
|--------------|-----------------------------|
| <i>base</i>  | Base address of GPV address |
| <i>value</i> | Target value (0 - 15)       |

**56.3.2 static nic\_qos\_t NIC\_GetReadQos ( nic\_reg\_t *base* ) [inline], [static]**

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | Base address of GPV address |
|-------------|-----------------------------|

Returns

Current value configured

**56.3.3 static void NIC\_SetWriteQos ( nic\_reg\_t *base*, nic\_qos\_t *value* ) [inline], [static]**

Parameters

|              |                             |
|--------------|-----------------------------|
| <i>base</i>  | Base address of GPV address |
| <i>value</i> | Target value (0 - 15)       |

**56.3.4 static nic\_qos\_t NIC\_GetWriteQos ( nic\_reg\_t *base* ) [inline], [static]**

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | Base address of GPV address |
|-------------|-----------------------------|

Returns

Current value configured

**56.3.5 static void NIC\_SetFnModAhb ( nic\_reg\_t *base*, nic\_fn\_mod\_ahb\_t *v* ) [inline], [static]**

Parameters

|              |                             |
|--------------|-----------------------------|
| <i>base</i>  | Base address of GPV address |
| <i>value</i> | Target value                |

**56.3.6 static nic\_fn\_mod\_ahb\_t NIC\_GetFnModAhb ( nic\_reg\_t *base* ) [inline], [static]**

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | Base address of GPV address |
|-------------|-----------------------------|

Returns

Current value configured

**56.3.7 static void NIC\_SetWrTideMark ( nic\_reg\_t *base*, uint8\_t *value* ) [inline], [static]**

Parameters

|              |                             |
|--------------|-----------------------------|
| <i>base</i>  | Base address of GPV address |
| <i>value</i> | Target value (0 - 7)        |

**56.3.8 static uint8\_t NIC\_GetWrTideMark ( nic\_reg\_t *base* ) [inline], [static]**

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | Base address of GPV address |
|-------------|-----------------------------|

Returns

Current value configured

**56.3.9 static void NIC\_SetFnMod ( nic\_reg\_t *base*, nic\_fn\_mod\_t *value* ) [inline], [static]**

Parameters

|              |                             |
|--------------|-----------------------------|
| <i>base</i>  | Base address of GPV address |
| <i>value</i> | Target value                |

**56.3.10** `static nic_fn_mod_t NIC_GetFnMod ( nic_reg_t base ) [inline], [static]`

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | Base address of GPV address |
|-------------|-----------------------------|

Returns

Current value configured

**56.3.11** `static void NIC_SetFnMod2 ( nic_reg_t base, nic_fn_mod_t value ) [inline], [static]`

Parameters

|              |                             |
|--------------|-----------------------------|
| <i>base</i>  | Base address of GPV address |
| <i>value</i> | Target value                |

**56.3.12** `static nic_fn_mod2_t NIC_GetFnMod2 ( nic_reg_t base ) [inline], [static]`

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | Base address of GPV address |
|-------------|-----------------------------|

Returns

Current value configured



## 56.3.13 CODEC Adapter

### 56.3.13.1 Overview

#### Enumerations

- enum {
  - [kCODEC\\_WM8904](#),
  - [kCODEC\\_WM8960](#),
  - [kCODEC\\_WM8524](#),
  - [kCODEC\\_SGTL5000](#),
  - [kCODEC\\_DA7212](#),
  - [kCODEC\\_CS42888](#),
  - [kCODEC\\_CS42448](#),
  - [kCODEC\\_AK4497](#),
  - [kCODEC\\_AK4458](#),
  - [kCODEC\\_TFA9XXX](#),
  - [kCODEC\\_TFA9896](#),
  - [kCODEC\\_WM8962](#) }

*codec type*

### 56.3.13.2 Enumeration Type Documentation

#### 56.3.13.2.1 anonymous enum

Enumerator

***kCODEC\_WM8904*** wm8904  
***kCODEC\_WM8960*** wm8960  
***kCODEC\_WM8524*** wm8524  
***kCODEC\_SGTL5000*** sgtl5000  
***kCODEC\_DA7212*** da7212  
***kCODEC\_CS42888*** CS42888.  
***kCODEC\_CS42448*** CS42448.  
***kCODEC\_AK4497*** AK4497.  
***kCODEC\_AK4458*** ak4458  
***kCODEC\_TFA9XXX*** tfa9xxx  
***kCODEC\_TFA9896*** tfa9896  
***kCODEC\_WM8962*** wm8962

**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.

© 2021 NXP B.V.

