

Document Number: MCUXSDKAPIRM
Rev 2.11.0
Jan 2022

MCUXpresso SDK API Reference Manual

NXP Semiconductors



Contents

Chapter 1 Introduction

Chapter 2 Trademarks

Chapter 3 Architectural Overview

Chapter 4 Clock Driver

4.1	Overview	7
4.2	Data Structure Documentation	17
4.2.1	struct clock_usb_pll_config_t	17
4.2.2	struct clock_sys_pll_config_t	17
4.2.3	struct clock_audio_pll_config_t	18
4.2.4	struct clock_enet_pll_config_t	18
4.3	Macro Definition Documentation	19
4.3.1	FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL	19
4.3.2	FSL_CLOCK_DRIVER_VERSION	19
4.3.3	ADC_CLOCKS	19
4.3.4	AOI_CLOCKS	19
4.3.5	DCDC_CLOCKS	20
4.3.6	DCP_CLOCKS	20
4.3.7	DMAMUX_CLOCKS	20
4.3.8	EDMA_CLOCKS	20
4.3.9	EWM_CLOCKS	20
4.3.10	FLEXIO_CLOCKS	21
4.3.11	FLEXRAM_CLOCKS	21
4.3.12	FLEXSPI_CLOCKS	21
4.3.13	GPIO_CLOCKS	21
4.3.14	GPT_CLOCKS	21
4.3.15	KPP_CLOCKS	22
4.3.16	LPI2C_CLOCKS	22
4.3.17	LPSPI_CLOCKS	22
4.3.18	LPUART_CLOCKS	22
4.3.19	OCRAM_EXSC_CLOCKS	22
4.3.20	PIT_CLOCKS	23
4.3.21	PWM_CLOCKS	23

Section No.	Title	Page No.
4.3.22	RTWDOG_CLOCKS	23
4.3.23	SAI_CLOCKS	23
4.3.24	TRNG_CLOCKS	23
4.3.25	WDOG_CLOCKS	24
4.3.26	SPDIF_CLOCKS	24
4.3.27	XBARA_CLOCKS	24
4.3.28	kCLOCK_CoreSysClk	24
4.3.29	CLOCK_GetCoreSysClkFreq	24
4.4	Enumeration Type Documentation	24
4.4.1	clock_name_t	24
4.4.2	clock_ip_name_t	25
4.4.3	clock_osc_t	26
4.4.4	clock_gate_value_t	27
4.4.5	clock_mode_t	27
4.4.6	clock_mux_t	27
4.4.7	clock_div_t	28
4.4.8	clock_div_value_t	28
4.4.9	clock_usb_src_t	32
4.4.10	clock_usb_phy_src_t	32
4.4.11	_clock_pll_clk_src	33
4.4.12	clock_pll_t	33
4.4.13	clock_pfd_t	33
4.4.14	clock_output1_selection_t	33
4.4.15	clock_output2_selection_t	34
4.4.16	clock_output_divider_t	34
4.4.17	clock_root_t	34
4.5	Function Documentation	35
4.5.1	CLOCK_SetMux	35
4.5.2	CLOCK_GetMux	36
4.5.3	CLOCK_SetDiv	36
4.5.4	CLOCK_GetDiv	36
4.5.5	CLOCK_ControlGate	37
4.5.6	CLOCK_EnableClock	37
4.5.7	CLOCK_DisableClock	37
4.5.8	CLOCK_SetMode	37
4.5.9	CLOCK_GetOscFreq	38
4.5.10	CLOCK_GetCoreFreq	38
4.5.11	CLOCK_GetIpgFreq	38
4.5.12	CLOCK_GetPerClkFreq	38
4.5.13	CLOCK_GetFreq	38
4.5.14	CLOCK_GetCpuClkFreq	39
4.5.15	CLOCK_GetClockRootFreq	39
4.5.16	CLOCK_InitExternalClk	39

Section No.	Title	Page No.
4.5.17	<code>CLOCK_DeinitExternalClk</code>	39
4.5.18	<code>CLOCK_SwitchOsc</code>	39
4.5.19	<code>CLOCK_GetRtcFreq</code>	40
4.5.20	<code>CLOCK_SetXtalFreq</code>	40
4.5.21	<code>CLOCK_SetRtcXtalFreq</code>	40
4.5.22	<code>CLOCK_EnableUsbhs0Clock</code>	40
4.6	Variable Documentation	41
4.6.1	<code>g_xtalFreq</code>	41
4.6.2	<code>g_rtcXtalFreq</code>	41

Chapter 5 IOMUXC: IOMUX Controller

5.1	Overview	42
5.2	Macro Definition Documentation	51
5.2.1	<code>FSL_IOMUXC_DRIVER_VERSION</code>	51
5.3	Function Documentation	51
5.3.1	<code>IOMUXC_SetPinMux</code>	51
5.3.2	<code>IOMUXC_SetPinConfig</code>	52
5.3.3	<code>IOMUXC_EnableMode</code>	52
5.3.4	<code>IOMUXC_SetSaiMClkClockSource</code>	53
5.3.5	<code>IOMUXC_MQSEEnterSoftwareReset</code>	53
5.3.6	<code>IOMUXC_MQSEnable</code>	53
5.3.7	<code>IOMUXC_MQSConfig</code>	53

Chapter 6 ADC_ETC: ADC External Trigger Control

6.1	Overview	55
6.2	Typical use case	55
6.2.1	<code>Software trigger Configuration</code>	55
6.2.2	<code>Hardware trigger Configuration</code>	55
6.3	Data Structure Documentation	56
6.3.1	<code>struct adc_etc_config_t</code>	56
6.3.2	<code>struct adc_etc_trigger_chain_config_t</code>	56
6.3.3	<code>struct adc_etc_trigger_config_t</code>	56
6.4	Macro Definition Documentation	56
6.4.1	<code>FSL_ADC_ETC_DRIVER_VERSION</code>	57
6.4.2	<code>ADC_ETC_DMA_CTRL_TRGn_REQ_MASK</code>	57
6.5	Function Documentation	57
6.5.1	<code>ADC_ETC_Init</code>	57

Section No.	Title	Page No.
6.5.2	ADC_ETC_Deinit	57
6.5.3	ADC_ETC_GetDefaultConfig	57
6.5.4	ADC_ETC_SetTriggerConfig	58
6.5.5	ADC_ETC_SetTriggerChainConfig	58
6.5.6	ADC_ETC_GetInterruptStatusFlags	58
6.5.7	ADC_ETC_ClearInterruptStatusFlags	59
6.5.8	ADC_ETC_EnableDMA	59
6.5.9	ADC_ETC_DisableDMA	59
6.5.10	ADC_ETC_GetDMAStatusFlags	59
6.5.11	ADC_ETC_ClearDMAStatusFlags	60
6.5.12	ADC_ETC_DoSoftwareReset	60
6.5.13	ADC_ETC_DoSoftwareTrigger	60
6.5.14	ADC_ETC_GetADCConversionValue	61

Chapter 7 AIPSTZ: AHB to IP Bridge

7.1	Overview	62
7.2	Enumeration Type Documentation	62
7.2.1	aipstz_master_privilege_level_t	62
7.2.2	aipstz_master_t	63
7.2.3	aipstz_peripheral_access_control_t	63
7.2.4	aipstz_peripheral_t	63
7.3	Function Documentation	63
7.3.1	AIPSTZ_SetMasterPriviledgeLevel	63
7.3.2	AIPSTZ_SetPeripheralAccessControl	63

Chapter 8 AOI: Crossbar AND/OR/INVERT Driver

8.1	Overview	65
8.2	Function groups	65
8.2.1	AOI Initialization	65
8.2.2	AOI Get Set Operation	65
8.3	Typical use case	65
8.4	Data Structure Documentation	67
8.4.1	struct aoi_event_config_t	67
8.5	Macro Definition Documentation	68
8.5.1	FSL_AOI_DRIVER_VERSION	68
8.6	Enumeration Type Documentation	68
8.6.1	aoi_input_config_t	68

Section No.	Title	Page No.
8.6.2	aoi_event_t	68
8.7	Function Documentation	68
8.7.1	AOI_Init	68
8.7.2	AOI_Deinit	68
8.7.3	AOI_GetEventLogicConfig	69
8.7.4	AOI_SetEventLogicConfig	69

Chapter 9 CACHE: ARMV7-M7 CACHE Memory Controller

9.1	Overview	71
9.2	Function groups	71
9.2.1	L1 CACHE Operation	71
9.2.2	L2 CACHE Operation	71
9.3	Macro Definition Documentation	72
9.3.1	FSL_CACHE_DRIVER_VERSION	72
9.4	Function Documentation	72
9.4.1	L1CACHE_InvalidateICacheByRange	73
9.4.2	L1CACHE_InvalidateDCacheByRange	74
9.4.3	L1CACHE_CleanDCacheByRange	74
9.4.4	L1CACHE_CleanInvalidateDCacheByRange	75
9.4.5	ICACHE_InvalidateByRange	76
9.4.6	DCACHE_InvalidateByRange	76
9.4.7	DCACHE_CleanByRange	77
9.4.8	DCACHE_CleanInvalidateByRange	78

Chapter 10 Common Driver

10.1	Overview	79
10.2	Macro Definition Documentation	81
10.2.1	FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ	81
10.2.2	MAKE_STATUS	81
10.2.3	MAKE_VERSION	82
10.2.4	FSL_COMMON_DRIVER_VERSION	82
10.2.5	DEBUG_CONSOLE_DEVICE_TYPE_NONE	82
10.2.6	DEBUG_CONSOLE_DEVICE_TYPE_UART	82
10.2.7	DEBUG_CONSOLE_DEVICE_TYPE_LPUART	82
10.2.8	DEBUG_CONSOLE_DEVICE_TYPE_LPSCI	82
10.2.9	DEBUG_CONSOLE_DEVICE_TYPE_USBCDC	82
10.2.10	DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM	82
10.2.11	DEBUG_CONSOLE_DEVICE_TYPE_IUART	82

Section No.	Title	Page No.
10.2.12	DEBUG_CONSOLE_DEVICE_TYPE_VUSART	82
10.2.13	DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART	82
10.2.14	DEBUG_CONSOLE_DEVICE_TYPE_SWO	82
10.2.15	DEBUG_CONSOLE_DEVICE_TYPE_QSCI	82
10.2.16	ARRAY_SIZE	82
10.3	Typedef Documentation	82
10.3.1	status_t	82
10.4	Enumeration Type Documentation	83
10.4.1	_status_groups	83
10.4.2	anonymous enum	85
10.5	Function Documentation	85
10.5.1	SDK_Malloc	86
10.5.2	SDK_Free	87
10.5.3	SDK_DelayAtLeastUs	87
Chapter 11 DCDC: DCDC Converter		
11.1	Overview	88
11.2	Function groups	88
11.2.1	Initialization and deinitialization	88
11.2.2	Status	88
11.2.3	Misc control	88
11.3	Application guideline	89
11.3.1	Continous conduction mode	89
11.3.2	Discontinous conduction mode	89
11.4	Data Structure Documentation	91
11.4.1	struct dcdc_detection_config_t	91
11.4.2	struct dcdc_loop_control_config_t	92
11.4.3	struct dcdc_low_power_config_t	93
11.4.4	struct dcdc_internal_regulator_config_t	93
11.4.5	struct dcdc_min_power_config_t	94
11.5	Macro Definition Documentation	94
11.5.1	FSL_DCDC_DRIVER_VERSION	94
11.6	Enumeration Type Documentation	94
11.6.1	_dcdc_status_flags_t	94
11.6.2	dcdc_comparator_current_bias_t	94
11.6.3	dcdc_over_current_threshold_t	95
11.6.4	dcdc_peak_current_threshold_t	95

Section No.	Title	Page No.
11.6.5	<code>dcdc_count_charging_time_period_t</code>	95
11.6.6	<code>dcdc_count_charging_time_threshold_t</code>	95
11.6.7	<code>dcdc_clock_source_t</code>	95
11.7	Function Documentation	96
11.7.1	<code>DCDC_Init</code>	96
11.7.2	<code>DCDC_Deinit</code>	96
11.7.3	<code>DCDC_GetstatusFlags</code>	96
11.7.4	<code>DCDC_EnableOutputRangeComparator</code>	96
11.7.5	<code>DCDC_SetClockSource</code>	96
11.7.6	<code>DCDC_GetDefaultDetectionConfig</code>	97
11.7.7	<code>DCDC_SetDetectionConfig</code>	97
11.7.8	<code>DCDC_GetDefaultLowPowerConfig</code>	97
11.7.9	<code>DCDC_SetLowPowerConfig</code>	98
11.7.10	<code>DCDC_ResetCurrentAlertSignal</code>	98
11.7.11	<code>DCDC_SetBandgapVoltageTrimValue</code>	98
11.7.12	<code>DCDC_GetDefaultLoopControlConfig</code>	98
11.7.13	<code>DCDC_SetLoopControlConfig</code>	99
11.7.14	<code>DCDC_SetMinPowerConfig</code>	99
11.7.15	<code>DCDC_SetLPComparatorBias Value</code>	99
11.7.16	<code>DCDC_LockTargetVoltage</code>	99
11.7.17	<code>DCDC_AdjustTargetVoltage</code>	100
11.7.18	<code>DCDC_AdjustRunTargetVoltage</code>	100
11.7.19	<code>DCDC_AdjustLowPowerTargetVoltage</code>	101
11.7.20	<code>DCDC_SetInternalRegulatorConfig</code>	101
11.7.21	<code>DCDC_EnableImproveTransition</code>	101
11.7.22	<code>DCDC_BootIntoDCM</code>	101
11.7.23	<code>DCDC_BootIntoCCM</code>	102

Chapter 12 DCP: Data Co-Processor

12.1	Overview	103
12.2	DCP Driver Initialization and Configuration	103
12.3	Comments about API usage in RTOS	104
12.4	Comments about API usage in interrupt handler	104
12.5	Comments about DCACHE	104
12.6	DCP Driver Examples	104
12.6.1	Simple examples	104
12.7	Data Structure Documentation	106
12.7.1	<code>struct dcp_work_packet_t</code>	106

Section No.	Title	Page No.
12.7.2	struct dcp_handle_t	106
12.7.3	struct dcp_context_t	106
12.7.4	struct dcp_config_t	106
12.8	Macro Definition Documentation	107
12.8.1	FSL_DCP_DRIVER_VERSION	107
12.9	Enumeration Type Documentation	107
12.9.1	_dcp_status	108
12.9.2	_dcp_ch_enable_t	108
12.9.3	_dcp_ch_int_enable_t	108
12.9.4	dcp_channel_t	108
12.9.5	dcp_key_slot_t	108
12.9.6	dcp_swap_t	109
12.10	Function Documentation	109
12.10.1	DCP_Init	109
12.10.2	DCP_Deinit	109
12.10.3	DCP_GetDefaultConfig	109
12.10.4	DCP_WaitForChannelComplete	110
12.11	DCP AES blocking driver	111
12.11.1	Overview	111
12.11.2	Function Documentation	111
12.12	DCP AES non-blocking driver	115
12.12.1	Overview	115
12.12.2	Function Documentation	115
12.13	DCP HASH driver	119
12.13.1	Overview	119
12.13.2	Data Structure Documentation	120
12.13.3	Macro Definition Documentation	120
12.13.4	Enumeration Type Documentation	120
12.13.5	Function Documentation	120

Chapter 13 DMAMUX: Direct Memory Access Multiplexer Driver

13.1	Overview	123
13.2	Typical use case	123
13.2.1	DMAMUX Operation	123
13.3	Macro Definition Documentation	123
13.3.1	FSL_DMAMUX_DRIVER_VERSION	123

Section No.	Title	Page No.
13.4 Function Documentation		124
13.4.1 DMAMUX_Init		124
13.4.2 DMAMUX_Deinit		125
13.4.3 DMAMUX_EnableChannel		125
13.4.4 DMAMUX_DisableChannel		125
13.4.5 DMAMUX_SetSource		126
13.4.6 DMAMUX_EnablePeriodTrigger		126
13.4.7 DMAMUX_DisablePeriodTrigger		126
13.4.8 DMAMUX_EnableAlwaysOn		126

Chapter 14 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

14.1 Overview		128
14.2 Typical use case		128
14.2.1 eDMA Operation		128
14.3 Data Structure Documentation		133
14.3.1 struct edma_config_t		133
14.3.2 struct edma_transfer_config_t		134
14.3.3 struct edma_channel_Preemption_config_t		135
14.3.4 struct edma_minor_offset_config_t		135
14.3.5 struct edma_tcd_t		135
14.3.6 struct edma_handle_t		136
14.4 Macro Definition Documentation		137
14.4.1 FSL_EDMA_DRIVER_VERSION		137
14.5 Typedef Documentation		137
14.5.1 edma_callback		137
14.6 Enumeration Type Documentation		138
14.6.1 edma_transfer_size_t		138
14.6.2 edma_modulo_t		138
14.6.3 edma_bandwidth_t		139
14.6.4 edma_channel_link_type_t		139
14.6.5 anonymous enum		139
14.6.6 anonymous enum		140
14.6.7 edma_interrupt_enable_t		140
14.6.8 edma_transfer_type_t		140
14.6.9 anonymous enum		140
14.7 Function Documentation		141
14.7.1 EDMA_Init		141
14.7.2 EDMA_Deinit		142
14.7.3 EDMA_InstallTCD		142

Section No.	Title	Page No.
14.7.4	EDMA_GetDefaultConfig	142
14.7.5	EDMA_EnableContinuousChannelLinkMode	143
14.7.6	EDMA_EnableMinorLoopMapping	143
14.7.7	EDMA_ResetChannel	143
14.7.8	EDMA_SetTransferConfig	144
14.7.9	EDMA_SetMinorOffsetConfig	144
14.7.10	EDMA_SetChannelPreemptionConfig	145
14.7.11	EDMA_SetChannelLink	145
14.7.12	EDMA_SetBandWidth	146
14.7.13	EDMA_SetModulo	146
14.7.14	EDMA_EnableAsyncRequest	147
14.7.15	EDMA_EnableAutoStopRequest	147
14.7.16	EDMA_EnableChannelInterrupts	147
14.7.17	EDMA_DisableChannelInterrupts	147
14.7.18	EDMA_SetMajorOffsetConfig	148
14.7.19	EDMA_TcdReset	148
14.7.20	EDMA_TcdSetTransferConfig	148
14.7.21	EDMA_TcdSetMinorOffsetConfig	149
14.7.22	EDMA_TcdSetChannelLink	149
14.7.23	EDMA_TcdSetBandWidth	150
14.7.24	EDMA_TcdSetModulo	150
14.7.25	EDMA_TcdEnableAutoStopRequest	151
14.7.26	EDMA_TcdEnableInterrupts	152
14.7.27	EDMA_TcdDisableInterrupts	152
14.7.28	EDMA_TcdSetMajorOffsetConfig	152
14.7.29	EDMA_EnableChannelRequest	152
14.7.30	EDMA_DisableChannelRequest	153
14.7.31	EDMA_TriggerChannelStart	153
14.7.32	EDMA_GetRemainingMajorLoopCount	153
14.7.33	EDMA_GetErrorStatusFlags	154
14.7.34	EDMA_GetChannelStatusFlags	154
14.7.35	EDMA_ClearChannelStatusFlags	154
14.7.36	EDMA_CreateHandle	155
14.7.37	EDMA_InstallTCDMemory	155
14.7.38	EDMA_SetCallback	155
14.7.39	EDMA_PreparesTransferConfig	156
14.7.40	EDMA_PreparesTransfer	156
14.7.41	EDMA_SubmitTransfer	157
14.7.42	EDMA_StartTransfer	158
14.7.43	EDMA_StopTransfer	159
14.7.44	EDMA_AbortTransfer	159
14.7.45	EDMA_GetUnusedTCDNumber	159
14.7.46	EDMA_GetNextTCDAAddress	159
14.7.47	EDMA_HandleIRQ	160

Section No.	Title	Page No.
Chapter 15 EWM: External Watchdog Monitor Driver		
15.1	Overview	161
15.2	Typical use case	161
15.3	Data Structure Documentation	162
15.3.1	struct ewm_config_t	162
15.4	Macro Definition Documentation	162
15.4.1	FSL_EWM_DRIVER_VERSION	162
15.5	Enumeration Type Documentation	162
15.5.1	ewm_lpo_clock_source_t	162
15.5.2	_ewm_interrupt_enable_t	163
15.5.3	_ewm_status_flags_t	163
15.6	Function Documentation	163
15.6.1	EWM_Init	163
15.6.2	EWM_Deinit	163
15.6.3	EWM_GetDefaultConfig	164
15.6.4	EWM_EnableInterrupts	164
15.6.5	EWM_DisableInterrupts	164
15.6.6	EWM_GetStatusFlags	165
15.6.7	EWM_Refresh	165
Chapter 16 FlexIO: FlexIO Driver		
16.1	Overview	167
16.2	FlexIO Driver	168
16.2.1	Overview	168
16.2.2	Data Structure Documentation	172
16.2.3	Macro Definition Documentation	175
16.2.4	Typedef Documentation	175
16.2.5	Enumeration Type Documentation	175
16.2.6	Function Documentation	180
16.2.7	Variable Documentation	190
16.3	FlexIO Camera Driver	191
16.3.1	Overview	191
16.3.2	Typical use case	191
16.3.3	Data Structure Documentation	194
16.3.4	Macro Definition Documentation	195
16.3.5	Enumeration Type Documentation	195
16.3.6	Function Documentation	196

Section No.	Title	Page No.
16.3.7	FlexIO eDMA Camera Driver	199
16.4	FlexIO I2C Master Driver	203
16.4.1	Overview	203
16.4.2	Typical use case	203
16.4.3	Data Structure Documentation	207
16.4.4	Macro Definition Documentation	209
16.4.5	Typedef Documentation	209
16.4.6	Enumeration Type Documentation	210
16.4.7	Function Documentation	210
16.5	FlexIO I2S Driver	220
16.5.1	Overview	220
16.5.2	Typical use case	220
16.5.3	Data Structure Documentation	225
16.5.4	Macro Definition Documentation	227
16.5.5	Enumeration Type Documentation	228
16.5.6	Function Documentation	229
16.5.7	FlexIO eDMA I2S Driver	240
16.6	FlexIO MCU Interface LCD Driver	246
16.6.1	Overview	246
16.6.2	Typical use case	246
16.6.3	Data Structure Documentation	252
16.6.4	Macro Definition Documentation	255
16.6.5	Typedef Documentation	256
16.6.6	Enumeration Type Documentation	256
16.6.7	Function Documentation	257
16.6.8	FlexIO eDMA MCU Interface LCD Driver	270
16.7	FlexIO SPI Driver	276
16.7.1	Overview	276
16.7.2	Typical use case	276
16.7.3	Data Structure Documentation	283
16.7.4	Macro Definition Documentation	286
16.7.5	Typedef Documentation	286
16.7.6	Enumeration Type Documentation	287
16.7.7	Function Documentation	288
16.7.8	FlexIO eDMA SPI Driver	302
16.8	FlexIO UART Driver	308
16.8.1	Overview	308
16.8.2	Typical use case	308
16.8.3	Data Structure Documentation	317
16.8.4	Macro Definition Documentation	319

Section No.	Title	Page No.
16.8.5	Typedef Documentation	319
16.8.6	Enumeration Type Documentation	320
16.8.7	Function Documentation	321
16.8.8	FlexIO eDMA UART Driver	332

Chapter 17 FLEXRAM: on-chip RAM manager

17.1	Overview	338
17.2	Macro Definition Documentation	340
17.2.1	FSL_FLEXRAM_DRIVER_VERSION	340
17.2.2	FLEXRAM_ECC_ERROR_DETAILED_INFO	340
17.3	Enumeration Type Documentation	340
17.3.1	anonymous enum	340
17.3.2	anonymous enum	340
17.3.3	flexram_tcm_access_mode_t	340
17.3.4	anonymous enum	341
17.4	Function Documentation	341
17.4.1	FLEXRAM_Init	341
17.4.2	FLEXRAM_GetInterruptStatus	341
17.4.3	FLEXRAM_ClearInterruptStatus	341
17.4.4	FLEXRAM_EnableInterruptStatus	341
17.4.5	FLEXRAM_DisableInterruptStatus	342
17.4.6	FLEXRAM_EnableInterruptSignal	342
17.4.7	FLEXRAM_DisableInterruptSignal	342
17.4.8	FLEXRAM_SetTCMReadAccessMode	342
17.4.9	FLEXRAM_SetTCMWriteAccessMode	343
17.4.10	FLEXRAM_EnableForceRamClockOn	343
17.4.11	FLEXRAM_SetOCRAMMagicAddr	343
17.4.12	FLEXRAM_SetDTCMMagicAddr	344
17.4.13	FLEXRAM_SetITCMMagicAddr	344

Chapter 18 FLEXSPI: Flexible Serial Peripheral Interface Driver

18.1	Overview	345
18.2	Data Structure Documentation	350
18.2.1	struct flexspi_config_t	350
18.2.2	struct flexspi_device_config_t	353
18.2.3	struct flexspi_transfer_t	354
18.2.4	struct _flexspi_handle	355
18.3	Macro Definition Documentation	355

Section No.	Title	Page No.
18.3.1	FSL_FLEXSPI_DRIVER_VERSION	355
18.3.2	FLEXSPI_LUT_SEQ	355
18.4	Typedef Documentation	356
18.4.1	flexspi_transfer_callback_t	356
18.5	Enumeration Type Documentation	356
18.5.1	anonymous enum	356
18.5.2	anonymous enum	356
18.5.3	flexspi_pad_t	357
18.5.4	flexspi_flags_t	357
18.5.5	flexspi_read_sample_clock_t	358
18.5.6	flexspi_cs_interval_cycle_unit_t	358
18.5.7	flexspi_ahb_write_wait_unit_t	358
18.5.8	flexspi_ip_error_code_t	359
18.5.9	flexspi_ahb_error_code_t	359
18.5.10	flexspi_port_t	359
18.5.11	flexspi_arb_command_source_t	359
18.5.12	flexspi_command_type_t	360
18.6	Function Documentation	360
18.6.1	FLEXSPIGetInstance	360
18.6.2	FLEXSPI_CheckAndClearError	360
18.6.3	FLEXSPI_Init	360
18.6.4	FLEXSPI_GetDefaultConfig	360
18.6.5	FLEXSPI_Deinit	361
18.6.6	FLEXSPI_UpdateDlValue	361
18.6.7	FLEXSPI_SetFlashConfig	361
18.6.8	FLEXSPI_SoftwareReset	362
18.6.9	FLEXSPI_Enable	363
18.6.10	FLEXSPI_EnableInterrupts	363
18.6.11	FLEXSPI_DisableInterrupts	363
18.6.12	FLEXSPI_EnableTxDMA	363
18.6.13	FLEXSPI_EnableRxDMA	364
18.6.14	FLEXSPI_GetTxFifoAddress	364
18.6.15	FLEXSPI_GetRxFifoAddress	364
18.6.16	FLEXSPI_ResetFifos	365
18.6.17	FLEXSPI_GetFifoCounts	366
18.6.18	FLEXSPI_GetInterruptStatusFlags	366
18.6.19	FLEXSPI_ClearInterruptStatusFlags	366
18.6.20	FLEXSPI_GetArbitratorCommandSource	367
18.6.21	FLEXSPI_GetIPCommandErrorCode	368
18.6.22	FLEXSPI_GetAHBCommandErrorCode	368
18.6.23	FLEXSPI_GetBusIdleStatus	368
18.6.24	FLEXSPI_UpdateRxSampleClock	369

Section No.	Title	Page No.
18.6.25	<code>FLEXSPI_EnableIPParallelMode</code>	369
18.6.26	<code>FLEXSPI_EnableAHBParallelMode</code>	369
18.6.27	<code>FLEXSPI_UpdateLUT</code>	369
18.6.28	<code>FLEXSPI_WriteData</code>	370
18.6.29	<code>FLEXSPI_ReadData</code>	370
18.6.30	<code>FLEXSPI_WriteBlocking</code>	370
18.6.31	<code>FLEXSPI_ReadBlocking</code>	371
18.6.32	<code>FLEXSPI_TransferBlocking</code>	372
18.6.33	<code>FLEXSPI_TransferCreateHandle</code>	372
18.6.34	<code>FLEXSPI_TransferNonBlocking</code>	373
18.6.35	<code>FLEXSPI_TransferGetCount</code>	373
18.6.36	<code>FLEXSPI_TransferAbort</code>	374
18.6.37	<code>FLEXSPI_TransferHandleIRQ</code>	374

Chapter 19 GPC: General Power Controller Driver

19.1	Overview	375
19.2	Macro Definition Documentation	375
19.2.1	<code>FSL_GPC_DRIVER_VERSION</code>	375
19.3	Function Documentation	375
19.3.1	<code>GPC_EnableIRQ</code>	375
19.3.2	<code>GPC_DisableIRQ</code>	376
19.3.3	<code>GPC_GetIRQStatusFlag</code>	376
19.3.4	<code>GPC_RequestPdram0PowerDown</code>	376
19.3.5	<code>GPC_RequestMEGAPowerOn</code>	377

Chapter 20 GPT: General Purpose Timer

20.1	Overview	378
20.2	Function groups	378
20.2.1	Initialization and deinitialization	378
20.3	Typical use case	378
20.3.1	GPT interrupt example	378
20.4	Data Structure Documentation	381
20.4.1	<code>struct gpt_config_t</code>	381
20.5	Enumeration Type Documentation	382
20.5.1	<code>gpt_clock_source_t</code>	382
20.5.2	<code>gpt_input_capture_channel_t</code>	382
20.5.3	<code>gpt_input_operation_mode_t</code>	383
20.5.4	<code>gpt_output_compare_channel_t</code>	383

Section No.	Title	Page No.
20.5.5	<code>gpt_output_operation_mode_t</code>	383
20.5.6	<code>gpt_interrupt_enable_t</code>	383
20.5.7	<code>gpt_status_flag_t</code>	384
20.6	Function Documentation	384
20.6.1	<code>GPT_Init</code>	384
20.6.2	<code>GPT_Deinit</code>	384
20.6.3	<code>GPT_GetDefaultConfig</code>	384
20.6.4	<code>GPT_SoftwareReset</code>	385
20.6.5	<code>GPT_SetClockSource</code>	385
20.6.6	<code>GPT_GetClockSource</code>	385
20.6.7	<code>GPT_SetClockDivider</code>	385
20.6.8	<code>GPT_GetClockDivider</code>	386
20.6.9	<code>GPT_SetOscClockDivider</code>	386
20.6.10	<code>GPT_GetOscClockDivider</code>	386
20.6.11	<code>GPT_StartTimer</code>	386
20.6.12	<code>GPT_StopTimer</code>	387
20.6.13	<code>GPT_GetCurrentTimerCount</code>	387
20.6.14	<code>GPT_SetInputOperationMode</code>	387
20.6.15	<code>GPT_GetInputOperationMode</code>	387
20.6.16	<code>GPT_GetInputCaptureValue</code>	388
20.6.17	<code>GPT_SetOutputOperationMode</code>	388
20.6.18	<code>GPT_GetOutputOperationMode</code>	389
20.6.19	<code>GPT_SetOutputCompareValue</code>	389
20.6.20	<code>GPT_GetOutputCompareValue</code>	389
20.6.21	<code>GPT_ForceOutput</code>	390
20.6.22	<code>GPT_EnableInterrupts</code>	390
20.6.23	<code>GPT_DisableInterrupts</code>	390
20.6.24	<code>GPT_GetEnabledInterrupts</code>	390
20.6.25	<code>GPT_GetStatusFlags</code>	391
20.6.26	<code>GPT_ClearStatusFlags</code>	391

Chapter 21 GPIO: General-Purpose Input/Output Driver

21.1	Overview	392
21.2	Typical use case	392
21.2.1	<code>Input Operation</code>	392
21.3	Data Structure Documentation	394
21.3.1	<code>struct gpio_pin_config_t</code>	394
21.4	Macro Definition Documentation	394
21.4.1	<code>FSL_GPIO_DRIVER_VERSION</code>	394

Section No.	Title	Page No.
21.5 Enumeration Type Documentation		394
21.5.1 <code>gpio_pin_direction_t</code>		394
21.5.2 <code>gpio_interrupt_mode_t</code>		394
21.6 Function Documentation		395
21.6.1 <code>GPIO_PinInit</code>		395
21.6.2 <code>GPIO_PinWrite</code>		396
21.6.3 <code>GPIO_WritePinOutput</code>		396
21.6.4 <code>GPIO_PortSet</code>		396
21.6.5 <code>GPIO_SetPinsOutput</code>		396
21.6.6 <code>GPIO_PortClear</code>		397
21.6.7 <code>GPIO_ClearPinsOutput</code>		398
21.6.8 <code>GPIO_PortToggle</code>		398
21.6.9 <code>GPIO_PinRead</code>		398
21.6.10 <code>GPIO_ReadPinInput</code>		398
21.6.11 <code>GPIO_PinReadPadStatus</code>		399
21.6.12 <code>GPIO_ReadPadStatus</code>		400
21.6.13 <code>GPIO_PinSetInterruptConfig</code>		400
21.6.14 <code>GPIO_SetPinInterruptConfig</code>		400
21.6.15 <code>GPIO_PortEnableInterrupts</code>		400
21.6.16 <code>GPIO_EnableInterrupts</code>		401
21.6.17 <code>GPIO_PortDisableInterrupts</code>		401
21.6.18 <code>GPIO_DisableInterrupts</code>		401
21.6.19 <code>GPIO_PortGetInterruptFlags</code>		401
21.6.20 <code>GPIO_GetPinsInterruptFlags</code>		402
21.6.21 <code>GPIO_PortClearInterruptFlags</code>		402
21.6.22 <code>GPIO_ClearPinsInterruptFlags</code>		402

Chapter 22 KPP: KeyPad Port Driver

22.1 Overview		404
22.2 Typical use case		404
22.3 Data Structure Documentation		405
22.3.1 <code>struct kpp_config_t</code>		405
22.4 Macro Definition Documentation		405
22.4.1 <code>FSL_KPP_DRIVER_VERSION</code>		405
22.5 Enumeration Type Documentation		406
22.5.1 <code>kpp_interrupt_enable_t</code>		406
22.5.2 <code>kpp_sync_operation_t</code>		406
22.6 Function Documentation		406
22.6.1 <code>KPP_Init</code>		406

Section No.	Title	Page No.
22.6.2	KPP_Deinit	406
22.6.3	KPP_EnableInterrupts	406
22.6.4	KPP_DisableInterrupts	407
22.6.5	KPP_GetStatusFlag	407
22.6.6	KPP_ClearStatusFlag	407
22.6.7	KPP_SetSynchronizeChain	408
22.6.8	KPP_KeyPressScanning	409

Chapter 23 LPI2C: Low Power Inter-Integrated Circuit Driver

23.1	Overview	410
23.2	Macro Definition Documentation	410
23.2.1	FSL_LPI2C_DRIVER_VERSION	410
23.2.2	I2C_RETRY_TIMES	411
23.3	Enumeration Type Documentation	411
23.3.1	anonymous enum	411
23.4	LPI2C Master Driver	412
23.4.1	Overview	412
23.4.2	Data Structure Documentation	415
23.4.3	Typedef Documentation	419
23.4.4	Enumeration Type Documentation	420
23.4.5	Function Documentation	422
23.5	LPI2C Slave Driver	436
23.5.1	Overview	436
23.5.2	Data Structure Documentation	438
23.5.3	Typedef Documentation	441
23.5.4	Enumeration Type Documentation	443
23.5.5	Function Documentation	444
23.6	LPI2C Master DMA Driver	453
23.6.1	Overview	453
23.6.2	Data Structure Documentation	453
23.6.3	Typedef Documentation	454
23.6.4	Function Documentation	456
23.7	LPI2C FreeRTOS Driver	459
23.7.1	Overview	459
23.7.2	Macro Definition Documentation	459
23.7.3	Function Documentation	459
23.8	LPI2C CMSIS Driver	462
23.8.1	LPI2C CMSIS Driver	462

Section No.	Title	Page No.
Chapter 24 LPSPI: Low Power Serial Peripheral Interface		
24.1	Overview	464
24.2	LPSPI Peripheral driver	465
24.2.1	Overview	465
24.2.2	Function groups	465
24.2.3	Typical use case	465
24.2.4	Data Structure Documentation	472
24.2.5	Macro Definition Documentation	478
24.2.6	Typedef Documentation	478
24.2.7	Enumeration Type Documentation	479
24.2.8	Function Documentation	484
24.2.9	Variable Documentation	499
24.3	LPSPI eDMA Driver	500
24.3.1	Overview	500
24.3.2	Data Structure Documentation	501
24.3.3	Macro Definition Documentation	505
24.3.4	Typedef Documentation	505
24.3.5	Function Documentation	505
24.4	LPSPI FreeRTOS Driver	511
24.4.1	Overview	511
24.4.2	Macro Definition Documentation	511
24.4.3	Function Documentation	511
24.5	LPSPI CMSIS Driver	514
24.5.1	Function groups	514
24.5.2	Typical use case	515
Chapter 25 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver		
25.1	Overview	516
25.2	LPUART Driver	517
25.2.1	Overview	517
25.2.2	Typical use case	517
25.2.3	Data Structure Documentation	522
25.2.4	Macro Definition Documentation	525
25.2.5	Typedef Documentation	525
25.2.6	Enumeration Type Documentation	525
25.2.7	Function Documentation	528
25.3	LPUART eDMA Driver	545
25.3.1	Overview	545

Section No.	Title	Page No.
25.3.2	Data Structure Documentation	546
25.3.3	Macro Definition Documentation	546
25.3.4	Typedef Documentation	546
25.3.5	Function Documentation	547
25.4	LPUART FreeRTOS Driver	551
25.4.1	Overview	551
25.4.2	Data Structure Documentation	551
25.4.3	Macro Definition Documentation	552
25.4.4	Function Documentation	552
25.5	LPUART CMSIS Driver	554
25.5.1	Function groups	554
Chapter 26 OCOTP: On Chip One-Time Programmable controller.		
26.1	Overview	556
26.2	OCOTP function group	556
26.2.1	Initialization and de-initialization	556
26.2.2	Read and Write operation	556
26.3	OCOTP example	556
26.4	Data Structure Documentation	557
26.4.1	struct ocotp_timing_t	557
26.5	Macro Definition Documentation	558
26.5.1	FSL_OCOTP_DRIVER_VERSION	558
26.6	Enumeration Type Documentation	558
26.6.1	anonymous enum	558
26.7	Function Documentation	558
26.7.1	OCOTP_Init	558
26.7.2	OCOTP_Deinit	559
26.7.3	OCOTP_CheckBusyStatus	559
26.7.4	OCOTP_CheckErrorStatus	559
26.7.5	OCOTP_ClearErrorStatus	560
26.7.6	OCOTP_ReloadShadowRegister	560
26.7.7	OCOTP_ReadFuseShadowRegister	560
26.7.8	OCOTP_ReadFuseShadowRegisterExt	561
26.7.9	OCOTP_WriteFuseShadowRegister	561
26.7.10	OCOTP_WriteFuseShadowRegisterWithLock	562
26.7.11	OCOTP_GetVersion	562

Section No.	Title	Page No.
Chapter 27 OTFAD: On The Fly AES-128 Decryption Driver		
27.1	Overview	564
27.2	Data Structure Documentation	565
27.2.1	struct otfad_encryption_config_t	565
27.2.2	struct otfad_config_t	566
27.3	Macro Definition Documentation	566
27.3.1	FSL_OTFAD_DRIVER_VERSION	566
27.4	Enumeration Type Documentation	566
27.4.1	anonymous enum	566
27.4.2	anonymous enum	566
27.4.3	anonymous enum	567
27.5	Function Documentation	567
27.5.1	OTFAD_GetDefaultConfig	567
27.5.2	OTFAD_Init	567
27.5.3	OTFAD_GetOperateMode	567
27.5.4	OTFAD_GetStatus	567
27.5.5	OTFAD_SetEncryptionConfig	568
27.5.6	OTFAD_GetEncryptionConfig	568
27.5.7	OTFAD_HitDetermination	568
Chapter 28 PIT: Periodic Interrupt Timer		
28.1	Overview	569
28.2	Function groups	569
28.2.1	Initialization and deinitialization	569
28.2.2	Timer period Operations	569
28.2.3	Start and Stop timer operations	569
28.2.4	Status	570
28.2.5	Interrupt	570
28.3	Typical use case	570
28.3.1	PIT tick example	570
28.4	Data Structure Documentation	571
28.4.1	struct pit_config_t	571
28.5	Enumeration Type Documentation	572
28.5.1	pit_chnl_t	572
28.5.2	pit_interrupt_enable_t	572
28.5.3	pit_status_flags_t	572

Section No.	Title	Page No.
28.6 Function Documentation		572
28.6.1 PIT_Init		572
28.6.2 PIT_Deinit		573
28.6.3 PIT_GetDefaultConfig		573
28.6.4 PIT_SetTimerChainMode		573
28.6.5 PIT_EnableInterrupts		574
28.6.6 PIT_DisableInterrupts		574
28.6.7 PIT_GetEnabledInterrupts		574
28.6.8 PIT_GetStatusFlags		575
28.6.9 PIT_ClearStatusFlags		576
28.6.10 PIT_SetTimerPeriod		576
28.6.11 PIT_GetCurrentTimerCount		577
28.6.12 PIT_StartTimer		577
28.6.13 PIT_StopTimer		577
28.6.14 PIT_GetLifetimeTimerCount		578
Chapter 29 PMU: Power Management Unit		
29.1 Overview		579
29.2 Macro Definition Documentation		581
29.2.1 FSL_PMU_DRIVER_VERSION		581
29.3 Enumeration Type Documentation		581
29.3.1 anonymous enum		581
29.3.2 pmu_1p1_weak_reference_source_t		582
29.3.3 pmu_3p0_vbus_voltage_source_t		582
29.3.4 pmu_core_reg_voltage_ramp_rate_t		582
29.3.5 pmu_power_bandgap_t		582
29.4 Function Documentation		582
29.4.1 PMU_GetStatusFlags		582
29.4.2 PMU_1P1SetWeakReferenceSource		583
29.4.3 PMU_1P1EnableWeakRegulator		583
29.4.4 PMU_1P1SetRegulatorOutputVoltage		583
29.4.5 PMU_1P1SetBrownoutOffsetVoltage		584
29.4.6 PMU_1P1EnablePullDown		584
29.4.7 PMU_1P1EnableCurrentLimit		584
29.4.8 PMU_1P1EnableBrownout		584
29.4.9 PMU_1P1EnableOutput		585
29.4.10 PMU_3P0SetRegulatorOutputVoltage		585
29.4.11 PMU_3P0SetVBusVoltageSource		585
29.4.12 PMU_3P0SetBrownoutOffsetVoltage		586
29.4.13 PMU_3P0EnableCurrentLimit		586
29.4.14 PMU_3P0EnableBrownout		586

Section No.	Title	Page No.
29.4.15	PMU_3P0EnableOutput	586
29.4.16	PMU_2P5EnableWeakRegulator	587
29.4.17	PMU_2P5SetRegulatorOutputVoltage	587
29.4.18	PMU_2P5SetBrownoutOffsetVoltage	587
29.4.19	PMU_2P5EnablePullDown	588
29.4.20	PMU_2P1EnablePullDown	588
29.4.21	PMU_2P5EnableCurrentLimit	588
29.4.22	PMU_2P5nableBrownout	588
29.4.23	PMU_2P5EnableOutput	589
29.4.24	PMU_CoreEnableIncreaseGateDrive	590
29.4.25	PMU_CoreSetRegulatorVoltageRampRate	590
29.4.26	PMU_CoreSetSOCDomainVoltage	590
29.4.27	PMU_CoreSetARMCoreDomainVoltage	591

Chapter 30 PWM: Pulse Width Modulator

30.1	Overview	592
30.2	PWM: Pulse Width Modulator	592
30.2.1	Initialization and deinitialization	592
30.2.2	PWM Operations	592
30.2.3	Input capture operations	592
30.2.4	Fault operation	592
30.2.5	PWM Start and Stop operations	593
30.2.6	Status	593
30.2.7	Interrupt	593
30.3	Register Update	593
30.4	Typical use case	593
30.4.1	PWM output	593
30.5	Data Structure Documentation	601
30.5.1	struct pwm_signal_param_t	601
30.5.2	struct pwm_config_t	601
30.5.3	struct pwm_fault_input_filter_param_t	602
30.5.4	struct pwm_fault_param_t	602
30.5.5	struct pwm_input_capture_param_t	602
30.6	Enumeration Type Documentation	603
30.6.1	pwm_submodule_t	603
30.6.2	pwm_value_register_t	603
30.6.3	_pwm_value_register_mask	603
30.6.4	pwm_clock_source_t	604
30.6.5	pwm_clock_prescale_t	604

Section No.	Title	Page No.
30.6.6	pwm_force_output_trigger_t	604
30.6.7	pwm_init_source_t	605
30.6.8	pwm_load_frequency_t	605
30.6.9	pwm_fault_input_t	605
30.6.10	pwm_fault_disable_t	606
30.6.11	pwm_input_capture_edge_t	606
30.6.12	pwm_force_signal_t	606
30.6.13	pwm_chnl_pair_operation_t	606
30.6.14	pwm_register_reload_t	607
30.6.15	pwm_fault_recovery_mode_t	607
30.6.16	pwm_interrupt_enable_t	607
30.6.17	pwm_status_flags_t	608
30.6.18	pwm_dma_enable_t	608
30.6.19	pwm_dma_source_select_t	608
30.6.20	pwm_watermark_control_t	609
30.6.21	pwm_mode_t	609
30.6.22	pwm_level_select_t	609
30.6.23	pwm_fault_state_t	609
30.6.24	pwm_reload_source_select_t	609
30.6.25	pwm_fault_clear_t	610
30.6.26	pwm_module_control_t	610
30.7	Function Documentation	610
30.7.1	PWM_Init	610
30.7.2	PWM_Deinit	610
30.7.3	PWM_GetDefaultConfig	611
30.7.4	PWM_SetupPwm	611
30.7.5	PWM_UpdatePwmDutyCycle	612
30.7.6	PWM_UpdatePwmDutyCycleHighAccuracy	612
30.7.7	PWM_SetupInputCapture	613
30.7.8	PWM_SetupFaultInputFilter	613
30.7.9	PWM_SetupFaults	613
30.7.10	PWM_FaultDefaultConfig	614
30.7.11	PWM_SetupForceSignal	614
30.7.12	PWM_EnableInterrupts	614
30.7.13	PWM_DisableInterrupts	615
30.7.14	PWM_GetEnabledInterrupts	615
30.7.15	PWM_DMAFIFOWatermarkControl	615
30.7.16	PWM_DMACaptureSourceSelect	616
30.7.17	PWM_EnableDMACapture	616
30.7.18	PWM_EnableDMAWrite	616
30.7.19	PWM_GetStatusFlags	617
30.7.20	PWM_ClearStatusFlags	617
30.7.21	PWM_StartTimer	617
30.7.22	PWM_StopTimer	618

Section No.	Title	Page No.
30.7.23	PWM_OutputTriggerEnable	618
30.7.24	PWM_ActivateOutputTrigger	618
30.7.25	PWM_DeactivateOutputTrigger	619
30.7.26	PWM_SetupSwCtrlOut	619
30.7.27	PWM_SetPwmLdok	620
30.7.28	PWM_SetPwmFaultState	620
30.7.29	PWM_SetupFaultDisableMap	620

Chapter 31 RTWDOG: 32-bit Watchdog Timer

31.1	Overview	622
31.2	Typical use case	622
31.3	Data Structure Documentation	624
31.3.1	struct rtwdog_work_mode_t	624
31.3.2	struct rtwdog_config_t	624
31.4	Macro Definition Documentation	625
31.4.1	FSL_RTWDOG_DRIVER_VERSION	625
31.5	Enumeration Type Documentation	625
31.5.1	rtwdog_clock_source_t	625
31.5.2	rtwdog_clock_prescaler_t	625
31.5.3	rtwdog_test_mode_t	625
31.5.4	_rtwdog_interrupt_enable_t	625
31.5.5	_rtwdog_status_flags_t	626
31.6	Function Documentation	626
31.6.1	RTWDOG_GetDefaultConfig	626
31.6.2	RTWDOG_Init	626
31.6.3	RTWDOG_Deinit	627
31.6.4	RTWDOG_Enable	627
31.6.5	RTWDOG_Disable	627
31.6.6	RTWDOG_EnableInterrupts	628
31.6.7	RTWDOG_DisableInterrupts	628
31.6.8	RTWDOG_GetStatusFlags	628
31.6.9	RTWDOG_EnableWindowMode	629
31.6.10	RTWDOG_CountToMesec	629
31.6.11	RTWDOG_ClearStatusFlags	629
31.6.12	RTWDOG_SetTimeoutValue	630
31.6.13	RTWDOG_SetWindowValue	630
31.6.14	RTWDOG_Unlock	630
31.6.15	RTWDOG_Refresh	631
31.6.16	RTWDOG_GetCounterValue	631

Section No.	Title	Page No.
Chapter 32 SAI: Serial Audio Interface		
32.1 Overview	632
32.2 Typical configurations	632
32.3 Typical use case	633
32.3.1 SAI Send/receive using an interrupt method	633
32.3.2 SAI Send/receive using a DMA method	633
32.4 SAI Driver	634
32.4.1 Overview	634
32.4.2 Data Structure Documentation	642
32.4.3 Macro Definition Documentation	645
32.4.4 Enumeration Type Documentation	646
32.4.5 Function Documentation	650
32.5 SAI EDMA Driver	680
32.5.1 Overview	680
32.5.2 Data Structure Documentation	681
32.5.3 Function Documentation	682
Chapter 33 SNVS: Secure Non-Volatile Storage		
33.1 Overview	693
33.2 Secure Non-Volatile Storage High-Power	694
33.2.1 Overview	694
33.2.2 Data Structure Documentation	697
33.2.3 Macro Definition Documentation	698
33.2.4 Enumeration Type Documentation	698
33.2.5 Function Documentation	700
33.3 Secure Non-Volatile Storage Low-Power	712
33.3.1 Overview	712
33.3.2 Data Structure Documentation	715
33.3.3 Enumeration Type Documentation	716
33.3.4 Function Documentation	716
Chapter 34 SPDIF: Sony/Philips Digital Interface		
34.1 Overview	724
34.2 Typical use case	724
34.2.1 SPDIF Send/receive using an interrupt method	724
34.2.2 SPDIF Send/receive using a DMA method	724

Section No.	Title	Page No.
34.3 Data Structure Documentation		729
34.3.1 struct spdif_config_t		729
34.3.2 struct spdif_transfer_t		730
34.3.3 struct _spdif_handle		730
34.4 Macro Definition Documentation		730
34.4.1 SPDIF_XFER_QUEUE_SIZE		730
34.5 Enumeration Type Documentation		730
34.5.1 anonymous enum		731
34.5.2 spdif_rxfull_select_t		731
34.5.3 spdif_txempty_select_t		731
34.5.4 spdif_uchannel_source_t		732
34.5.5 spdif_gain_select_t		732
34.5.6 spdif_tx_source_t		732
34.5.7 spdif_validity_config_t		732
34.5.8 anonymous enum		732
34.5.9 anonymous enum		733
34.6 Function Documentation		733
34.6.1 SPDIF_Init		733
34.6.2 SPDIF_GetDefaultConfig		734
34.6.3 SPDIF_Deinit		734
34.6.4 SPDIF.GetInstance		734
34.6.5 SPDIF_TxFIFOReset		734
34.6.6 SPDIF_RxFIFOReset		735
34.6.7 SPDIF_TxEnable		735
34.6.8 SPDIF_RxEnable		735
34.6.9 SPDIF_GetStatusFlag		735
34.6.10 SPDIF_ClearStatusFlags		736
34.6.11 SPDIF_EnableInterrupts		736
34.6.12 SPDIF_DisableInterrupts		736
34.6.13 SPDIF_EnableDMA		737
34.6.14 SPDIF_TxGetLeftDataRegisterAddress		737
34.6.15 SPDIF_TxGetRightDataRegisterAddress		738
34.6.16 SPDIF_RxGetLeftDataRegisterAddress		739
34.6.17 SPDIF_RxGetRightDataRegisterAddress		739
34.6.18 SPDIF_TxSetSampleRate		739
34.6.19 SPDIF_GetRxSampleRate		740
34.6.20 SPDIF_WriteBlocking		740
34.6.21 SPDIF_WriteLeftData		740
34.6.22 SPDIF_WriteRightData		741
34.6.23 SPDIF_WriteChannelStatusHigh		741
34.6.24 SPDIF_WriteChannelStatusLow		741
34.6.25 SPDIF_ReadBlocking		741

Section No.	Title	Page No.
34.6.26	SPDIF_ReadLeftData	742
34.6.27	SPDIF_ReadRightData	742
34.6.28	SPDIF_ReadChannelStatusHigh	742
34.6.29	SPDIF_ReadChannelStatusLow	743
34.6.30	SPDIF_ReadQChannel	744
34.6.31	SPDIF_ReadUChannel	744
34.6.32	SPDIF_TransferTxCreateHandle	744
34.6.33	SPDIF_TransferRxCreateHandle	745
34.6.34	SPDIF_TransferSendNonBlocking	745
34.6.35	SPDIF_TransferReceiveNonBlocking	746
34.6.36	SPDIF_TransferGetSendCount	746
34.6.37	SPDIF_TransferGetReceiveCount	747
34.6.38	SPDIF_TransferAbortSend	747
34.6.39	SPDIF_TransferAbortReceive	747
34.6.40	SPDIF_TransferTxHandleIRQ	748
34.6.41	SPDIF_TransferRxHandleIRQ	748
34.7	SPDIF eDMA Driver	749
34.7.1	Overview	749
34.7.2	Data Structure Documentation	750
34.7.3	Function Documentation	751

Chapter 35 SRC: System Reset Controller Driver

35.1	Overview	756
35.2	Macro Definition Documentation	757
35.2.1	FSL_SRC_DRIVER_VERSION	757
35.3	Enumeration Type Documentation	757
35.3.1	_src_reset_status_flags	757
35.3.2	src_warm_reset_bypass_count_t	758
35.4	Function Documentation	758
35.4.1	SRC_EnableWDOG3Reset	758
35.4.2	SRC_EnableCoreDebugResetAfterPowerGate	758
35.4.3	SRC_DoSoftwareResetARMCore0	758
35.4.4	SRC_GetSoftwareResetARMCore0Done	759
35.4.5	SRC_EnableWDOGReset	759
35.4.6	SRC_EnableLockupReset	759
35.4.7	SRC_GetBootModeWord1	760
35.4.8	SRC_GetBootModeWord2	761
35.4.9	SRC_GetResetStatusFlags	761
35.4.10	SRC_ClearResetStatusFlags	761
35.4.11	SRC_SetGeneralPurposeRegister	762

Section No.	Title	Page No.
35.4.12	SRC_GetGeneralPurposeRegister	762

Chapter 36 TEMPMON: Temperature Monitor Module

36.1	Overview	763
36.2	TEMPMON: Temperature Monitor Module	763
36.2.1	TEMPMON Operations	763
36.3	Data Structure Documentation	764
36.3.1	struct tempmon_config_t	764
36.4	Macro Definition Documentation	764
36.4.1	FSL_TEMPMON_DRIVER_VERSION	764
36.5	Enumeration Type Documentation	764
36.5.1	tempmon_alarm_mode	765
36.6	Function Documentation	765
36.6.1	TEMPMON_Init	765
36.6.2	TEMPMON_Deinit	765
36.6.3	TEMPMON_GetDefaultConfig	765
36.6.4	TEMPMON_StartMeasure	765
36.6.5	TEMPMON_StopMeasure	766
36.6.6	TEMPMON_GetCurrentTemperature	766
36.6.7	TEMPMON_SetTempAlarm	766

Chapter 37 TRNG: True Random Number Generator

37.1	Overview	767
37.2	TRNG Initialization	767
37.3	Get random data from TRNG	767
37.4	Data Structure Documentation	768
37.4.1	struct trng_statistical_check_limit_t	768
37.4.2	struct trng_config_t	769
37.5	Macro Definition Documentation	771
37.5.1	FSL_TRNG_DRIVER_VERSION	771
37.6	Enumeration Type Documentation	771
37.6.1	trng_sample_mode_t	772
37.6.2	trng_clock_mode_t	772
37.6.3	trng_ring_osc_div_t	772

Section No.	Title	Page No.
37.7 Function Documentation		772
37.7.1 TRNG_GetDefaultConfig		772
37.7.2 TRNG_Init		773
37.7.3 TRNG_Deinit		773
37.7.4 TRNG_GetRandomData		774

Chapter 38 WDOG: Watchdog Timer Driver

38.1 Overview		775
38.2 Typical use case		775
38.3 Data Structure Documentation		776
38.3.1 struct wdog_work_mode_t		776
38.3.2 struct wdog_config_t		776
38.4 Enumeration Type Documentation		777
38.4.1 _wdog_interrupt_enable		777
38.4.2 _wdog_status_flags		777
38.5 Function Documentation		777
38.5.1 WDOG_GetDefaultConfig		777
38.5.2 WDOG_Init		778
38.5.3 WDOG_Deinit		778
38.5.4 WDOG_Enable		778
38.5.5 WDOG_Disable		779
38.5.6 WDOG_TriggerSystemSoftwareReset		779
38.5.7 WDOG_TriggerSoftwareSignal		779
38.5.8 WDOG_EnableInterrupts		780
38.5.9 WDOG_GetStatusFlags		781
38.5.10 WDOG_ClearInterruptStatus		781
38.5.11 WDOG_SetTimeoutValue		782
38.5.12 WDOG_SetInterruptTimeoutValue		782
38.5.13 WDOG_DisablePowerDownEnable		782
38.5.14 WDOG_Refresh		783

Chapter 39 XBARA: Inter-Peripheral Crossbar Switch

39.1 Overview		784
39.2 Function		784
39.2.1 XBARA Initialization		784
39.2.2 Call diagram		784
39.3 Typical use case		784

Section No.	Title	Page No.
39.4 Data Structure Documentation		785
39.4.1 struct xbara_control_config_t		785
39.5 Enumeration Type Documentation		786
39.5.1 xbara_active_edge_t		786
39.5.2 xbara_request_t		786
39.5.3 xbara_status_flag_t		786
39.6 Function Documentation		787
39.6.1 XBARA_Init		787
39.6.2 XBARA_Deinit		788
39.6.3 XBARA_SetSignalsConnection		788
39.6.4 XBARA_GetStatusFlags		788
39.6.5 XBARA_ClearStatusFlags		789
39.6.6 XBARA_SetOutputSignalConfig		789
Chapter 40 Debug Console		
40.1 Overview		790
40.2 Function groups		790
40.2.1 Initialization		790
40.2.2 Advanced Feature		791
40.2.3 SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART		795
40.3 Typical use case		796
40.4 Macro Definition Documentation		798
40.4.1 DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN		798
40.4.2 DEBUGCONSOLE_REDIRECT_TO_SDK		798
40.4.3 DEBUGCONSOLE_DISABLE		798
40.4.4 SDK_DEBUGCONSOLE		798
40.4.5 PRINTF		798
40.5 Function Documentation		798
40.5.1 DbgConsole_Init		798
40.5.2 DbgConsole_Deinit		799
40.5.3 DbgConsole_EnterLowpower		799
40.5.4 DbgConsole_ExitLowpower		800
40.5.5 DbgConsole_Printf		800
40.5.6 DbgConsole_Vprintf		800
40.5.7 DbgConsole_Putchar		800
40.5.8 DbgConsole_Scanf		801
40.5.9 DbgConsole_Getchar		801
40.5.10 DbgConsole_BlockingPrintf		802
40.5.11 DbgConsole_BlockingVprintf		802

Section No.	Title	Page No.
40.5.12	DbgConsole_Flush	802
40.5.13	StrFormatPrintf	803
40.5.14	StrFormatScanf	803
40.6	Semihosting	804
40.6.1	Guide Semihosting for IAR	804
40.6.2	Guide Semihosting for Keil µVision	804
40.6.3	Guide Semihosting for MCUXpresso IDE	805
40.6.4	Guide Semihosting for ARMGCC	805

40.7	SWO	808
40.7.1	Guide SWO for SDK	808
40.7.2	Guide SWO for Keil µVision	809
40.7.3	Guide SWO for MCUXpresso IDE	810
40.7.4	Guide SWO for ARMGCC	810

Chapter 41 Notification Framework

41.1	Overview	811
41.2	Notifier Overview	811
41.3	Data Structure Documentation	813
41.3.1	struct notifier_notification_block_t	813
41.3.2	struct notifier_callback_config_t	814
41.3.3	struct notifier_handle_t	814
41.4	Typedef Documentation	815
41.4.1	notifier_user_config_t	815
41.4.2	notifier_user_function_t	815
41.4.3	notifier_callback_t	816
41.5	Enumeration Type Documentation	816
41.5.1	_notifier_status	816
41.5.2	notifier_policy_t	817
41.5.3	notifier_notification_type_t	817
41.5.4	notifier_callback_type_t	817
41.6	Function Documentation	817
41.6.1	NOTIFIER_CreateHandle	818
41.6.2	NOTIFIER_SwitchConfig	819
41.6.3	NOTIFIER_GetErrorCallbackIndex	820

Chapter 42 Shell

42.1	Overview	821
-------------	-----------------------	------------

Section No.	Title	Page No.
42.2	Function groups	821
42.2.1	Initialization	821
42.2.2	Advanced Feature	821
42.2.3	Shell Operation	821
42.3	Data Structure Documentation	823
42.3.1	struct shell_command_t	823
42.4	Macro Definition Documentation	824
42.4.1	SHELL_NON_BLOCKING_MODE	824
42.4.2	SHELL_AUTO_COMPLETE	824
42.4.3	SHELL_BUFFER_SIZE	824
42.4.4	SHELL_MAX_ARGS	824
42.4.5	SHELL_HISTORY_COUNT	824
42.4.6	SHELL_HANDLE_SIZE	824
42.4.7	SHELL_USE_COMMON_TASK	824
42.4.8	SHELL_TASK_PRIORITY	824
42.4.9	SHELL_TASK_STACK_SIZE	824
42.4.10	SHELL_HANDLE_DEFINE	825
42.4.11	SHELL_COMMAND_DEFINE	825
42.4.12	SHELL_COMMAND	826
42.5	Typedef Documentation	826
42.5.1	cmd_function_t	826
42.6	Enumeration Type Documentation	826
42.6.1	shell_status_t	826
42.7	Function Documentation	826
42.7.1	SHELL_Init	826
42.7.2	SHELL_RegisterCommand	827
42.7.3	SHELL_UnregisterCommand	828
42.7.4	SHELL_Write	828
42.7.5	SHELL_Printf	828
42.7.6	SHELL_WriteSynchronization	829
42.7.7	SHELL_PrintfSynchronization	829
42.7.8	SHELL_ChangePrompt	830
42.7.9	SHELL_PrintPrompt	830
42.7.10	SHELL_Task	830
42.7.11	SHELL_checkRunningInIsr	831

Chapter 43 CODEC Driver

43.1	Overview	832
43.2	CODEC Common Driver	833

Section No.	Title	Page No.
43.2.1	Overview	833
43.2.2	Data Structure Documentation	838
43.2.3	Macro Definition Documentation	839
43.2.4	Enumeration Type Documentation	839
43.2.5	Function Documentation	844
43.3	CODEC I2C Driver	848
43.3.1	Overview	848
43.3.2	Data Structure Documentation	849
43.3.3	Enumeration Type Documentation	849
43.3.4	Function Documentation	849
43.4	CS42888 Driver	852
43.4.1	Overview	852
43.4.2	Data Structure Documentation	854
43.4.3	Macro Definition Documentation	855
43.4.4	Enumeration Type Documentation	855
43.4.5	Function Documentation	856
43.4.6	CS42888 Adapter	862
43.5	DA7212 Driver	870
43.5.1	Overview	870
43.5.2	Data Structure Documentation	873
43.5.3	Macro Definition Documentation	874
43.5.4	Enumeration Type Documentation	874
43.5.5	Function Documentation	876
43.5.6	DA7212 Adapter	881
43.6	SGTL5000 Driver	889
43.6.1	Overview	889
43.6.2	Data Structure Documentation	891
43.6.3	Macro Definition Documentation	892
43.6.4	Enumeration Type Documentation	892
43.6.5	Function Documentation	894
43.6.6	SGTL5000 Adapter	900
43.7	WM8960 Driver	908
43.7.1	Overview	908
43.7.2	Data Structure Documentation	911
43.7.3	Macro Definition Documentation	913
43.7.4	Enumeration Type Documentation	913
43.7.5	Function Documentation	915
43.7.6	WM8960 Adapter	922
43.8	WM8904 Driver	930
43.8.1	Overview	930

Section No.	Title	Page No.
43.8.2	Data Structure Documentation	934
43.8.3	Macro Definition Documentation	935
43.8.4	Enumeration Type Documentation	935
43.8.5	Function Documentation	938
43.8.6	WM8904 Adapter	947

Chapter 44 Serial Manager

44.1	Overview	955
44.2	Data Structure Documentation	958
44.2.1	struct serial_manager_config_t	958
44.2.2	struct serial_manager_callback_message_t	958
44.3	Macro Definition Documentation	958
44.3.1	SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE	959
44.3.2	SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE	959
44.3.3	SERIAL_MANAGER_USE_COMMON_TASK	959
44.3.4	SERIAL_MANAGER_HANDLE_SIZE	959
44.3.5	SERIAL_MANAGER_HANDLE_DEFINE	959
44.3.6	SERIAL_MANAGER_WRITE_HANDLE_DEFINE	959
44.3.7	SERIAL_MANAGER_READ_HANDLE_DEFINE	960
44.3.8	SERIAL_MANAGER_TASK_PRIORITY	960
44.3.9	SERIAL_MANAGER_TASK_STACK_SIZE	960
44.4	Enumeration Type Documentation	960
44.4.1	serial_port_type_t	960
44.4.2	serial_manager_type_t	961
44.4.3	serial_manager_status_t	961
44.5	Function Documentation	961
44.5.1	SerialManager_Init	961
44.5.2	SerialManager_Deinit	962
44.5.3	SerialManager_OpenWriteHandle	963
44.5.4	SerialManager_CloseWriteHandle	964
44.5.5	SerialManager_OpenReadHandle	964
44.5.6	SerialManager_CloseReadHandle	965
44.5.7	SerialManager_WriteBlocking	966
44.5.8	SerialManager_ReadBlocking	966
44.5.9	SerialManager_EnterLowpower	967
44.5.10	SerialManager_ExitLowpower	967
44.5.11	SerialManager_needPollingIsr	968
44.6	Serial Port Uart	969
44.6.1	Overview	969

Section No.	Title	Page No.
44.6.2	Enumeration Type Documentation	969
44.7	Serial Port USB	970
44.7.1	Overview	970
44.7.2	Data Structure Documentation	970
44.7.3	Enumeration Type Documentation	971
44.7.4	USB Device Configuration	972
44.8	Serial Port SWO	973
44.8.1	Overview	973
44.8.2	Data Structure Documentation	973
44.8.3	Enumeration Type Documentation	973
44.8.4	CODEC Adapter	974

Chapter 1

Introduction

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

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

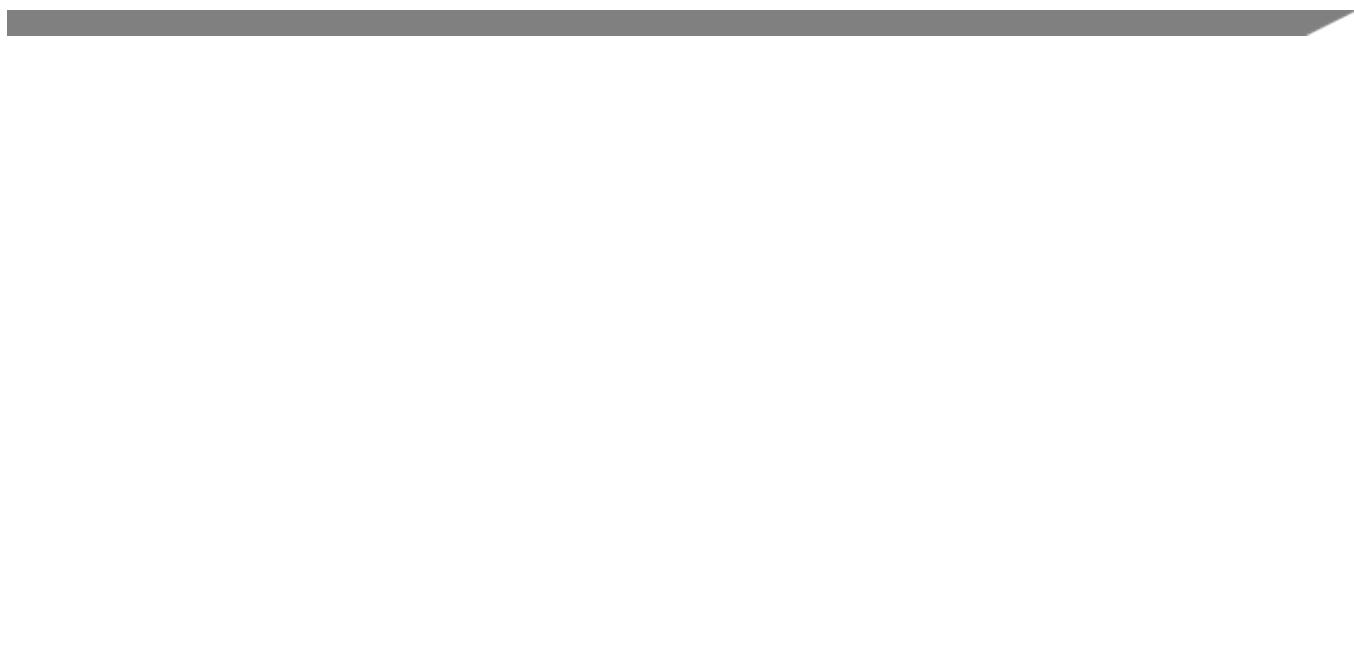
- Arm® and DSP standard libraries, and CMSIS-compliant device header files which provide direct access to the peripheral registers.
- Peripheral drivers that provide stateless, high-performance, ease-of-use APIs. Communication drivers provide higher-level transactional APIs for a higher-performance option.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

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

- IAR Embedded Workbench
- GNU Arm Embedded Toolchain

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

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



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

MCUXpresso SDK Folder Structure

Chapter 2

Trademarks

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

How to Reach Us:

Home Page: nxp.com

Web Support: nxp.com/support

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

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EM-BRACE, GREENCHIP, HITAG, I2C BUS,ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4M-OBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TD-MI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, Vision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2021 NXP B.V.

Chapter 3

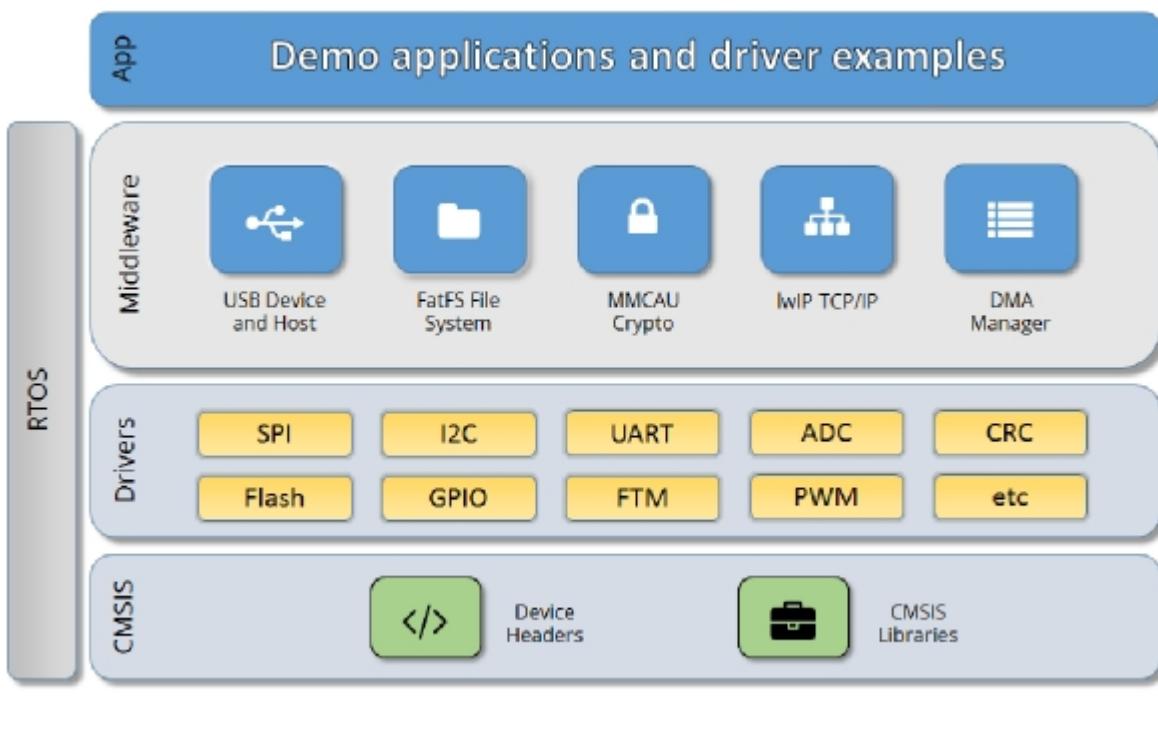
Architectural Overview

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

Overview

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

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



MCU header files

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

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

CMSIS Support

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

MCUXpresso SDK Peripheral Drivers

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

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

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

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

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

Interrupt handling for transactional APIs

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

```
PUBWEAK SPI0_IRQHandler  
PUBWEAK SPI0_DriverIRQHandler  
SPI0_IRQHandler
```

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

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

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

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

Feature Header Files

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

Application

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

Chapter 4

Clock Driver

4.1 Overview

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

The clock driver supports:

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

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)&0xFFFFU) << 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 DCDC_CLOCKS`
Clock ip name array for DCDC.
- `#define DCP_CLOCKS`

- `#define DMAMUX_CLOCKS`
Clock ip name array for DMAMUX.
- `#define EDMA_CLOCKS`
Clock ip name array for EDMA.
- `#define EWM_CLOCKS`
Clock ip name array for EWM.
- `#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 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 TRNG_CLOCKS`
Clock ip name array for TRNG.
- `#define WDOG_CLOCKS`
Clock ip name array for WDOG.
- `#define SPDIF_CLOCKS`
Clock ip name array for SPDIF.
- `#define XBARA_CLOCKS`
Clock ip name array for XBARA.
- `#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_CoreClk` = 0x1U,
 `kCLOCK_IpgClk` = 0x2U,
 `kCLOCK_PerClk` = 0x3U,
 `kCLOCK_OscClk` = 0x4U,
 `kCLOCK_RtcClk` = 0x5U,
 `kCLOCK_Usb1PllClk` = 0x6U,
 `kCLOCK_Usb1PllPfd0Clk` = 0x7U,
 `kCLOCK_Usb1PllPfd1Clk` = 0x8U,
 `kCLOCK_Usb1PllPfd2Clk` = 0x9U,
 `kCLOCK_Usb1PllPfd3Clk` = 0xAU,
 `kCLOCK_Usb1SwClk` = 0x12U,
 `kCLOCK_Usb1Sw60MClk` = 0x13U,
 `kCLOCK_Usb1Sw80MClk` = 0x14U,
 `kCLOCK_SysPllClk` = 0xBU,
 `kCLOCK_SysPllPfd0Clk` = 0xCU,
 `kCLOCK_SysPllPfd1Clk` = 0xDU,
 `kCLOCK_SysPllPfd2Clk` = 0xEU,
 `kCLOCK_SysPllPfd3Clk` = 0xFU,
 `kCLOCK_EnetPll500MClk` = 0x10U,
 `kCLOCK_AudioPllClk` = 0x11U,
 `kCLOCK_PерiphClk2` = 0x15U,
 `kCLOCK_FlexspiSel` = 0x16U,
 `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_FlexSpiExsc = (0U << 8U) | CCM_CCGR0(CG3_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_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_Lpssi1 = (1U << 8U) | CCM_CCGR1(CG0_SHIFT),
kCLOCK_Lpssi2 = (1U << 8U) | CCM_CCGR1(CG1_SHIFT),
kCLOCK_Pit = (1U << 8U) | CCM_CCGR1(CG6_SHIFT),
kCLOCK_Adcl = (1U << 8U) | CCM_CCGR1(CG8_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_OramExsc = (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_Ocotp = (2U << 8U) | CCM_CCGR2(CG6_SHIFT),
kCLOCK_Xbar1 = (2U << 8U) | CCM_CCGR2(CG11_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_IomuxcSnvsGpr = (3U << 8U) | CCM_CCGR3(CG15_SHIFT),
kCLOCK_Sim_m7_clk_r = (4U << 8U) | CCM_CCGR4(CG0_SHIFT),
kCLOCK_Iomuxc = (4U << 8U) | CCM_CCGR4(CG1_SHIFT),
kCLOCK_IomuxcGpr = (4U << 8U) | CCM_CCGR4(CG2_SHIFT),
kCLOCK_SimM7 = (4U << 8U) | CCM_CCGR4(CG4_SHIFT),
kCLOCK_SimM = (4U << 8U) | CCM_CCGR4(CG6_SHIFT),
kCLOCK_SimEms = (4U << 8U) | CCM_CCGR4(CG7_SHIFT),
kCLOCK_Pwm1 = (4U << 8U) | CCM_CCGR4(CG8_SHIFT),
kCLOCK_Dma_ps = (4U << 8U) | CCM_CCGR4(CG15_SHIFT),
kCLOCK_Rom = (5U << 8U) | CCM_CCGR5(CG0_SHIFT),
kCLOCK_Flexio1 = (5U << 8U) | CCM_CCGR5(CG1_SHIFT),
kCLOCK_Wdog3 = (5U << 8U) | CCM_CCGR5(CG2_SHIFT),
kCLOCK_Dma = (5U << 8U) | CCM_CCGR5(CG3_SHIFT),
kCLOCK_Kpp = (5U << 8U) | CCM_CCGR5(CG4_SHIFT),
kCLOCK_Wdog2 = (5U << 8U) | CCM_CCGR5(CG5_SHIFT)

```

MCUXpresso SDK API Reference Manual

- `kCLOCK_Anadig = (6U << 8U) | CCM_CCGR6_CG11_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_Pl3SwMux,`

`kCLOCK_PeriphMux,`

`kCLOCK_PrePeriphMux,`

`kCLOCK_TraceMux,`

`kCLOCK_PeriphClk2Mux,`

`kCLOCK_LpspiMux,`

`kCLOCK_FlexspiMux,`

`kCLOCK_FlexspiSrcMux,`

`kCLOCK_Sai3Mux,`

`kCLOCK_Sai1Mux,`

`kCLOCK_PerclkMux,`

`kCLOCK_Flexio1Mux,`

`kCLOCK_UartMux,`

`kCLOCK_SpdifMux,`

`kCLOCK_Lpi2cMux }`

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

```
kCLOCK_AhbDiv,  
kCLOCK_IpgDiv,  
kCLOCK_LpspiDiv,  
kCLOCK_FlexspiDiv,  
kCLOCK_PerclkDiv,  
kCLOCK_AdcDiv,  
kCLOCK_TraceDiv,  
kCLOCK_UartDiv,  
kCLOCK_Flexio1Div,  
kCLOCK_Sai3PreDiv,  
kCLOCK_Sai3Div,  
kCLOCK_Flexio1PreDiv,  
kCLOCK_Sai1PreDiv,  
kCLOCK_Sai1Div,  
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_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_LpspiDivBy9 = 8,
kCLOCK_LpspiDivBy10 = 9,
kCLOCK_LpspiDivBy11 = 10,
kCLOCK_LpspiDivBy12 = 11,
kCLOCK_LpspiDivBy13 = 12,
kCLOCK_LpspiDivBy14 = 13,
kCLOCK_LpspiDivBy15 = 14,
kCLOCK_LpspiDivBy16 = 15,
kCLOCK_FlexspiDivBy1 = 0,
kCLOCK_FlexspiDivBy2 = 1,
kCLOCK_FlexspiDivBy3 = 2,
kCLOCK_FlexspiDivBy4 = 3,
kCLOCK_FlexspiDivBy5 = 4,
kCLOCK_FlexspiDivBy6 = 5,
kCLOCK_FlexspiDivBy7 = 6,
kCLOCK_FlexspiDivBy8 = 7,
kCLOCK_AdcDivBy8 = 7,
kCLOCK_AdcDivBy12 = 11,
kCLOCK_AdcDivBy16 = 15,
kCLOCK_TraceDivBy1 = 0,
kCLOCK_TraceDivBy2 = 1,
kCLOCK_TraceDivBy3 = 2,
kCLOCK_TraceDivBy4 = 3,
kCLOCK_TraceDivBy5 = 4,
kCLOCK_TraceDivBy6 = 5,
kCLOCK_TraceDivBy7 = 6,
kCLOCK_TraceDivBy8 = 7,
kCLOCK_TraceDivBy9 = 8,
kCLOCK_TraceDivBy10 = 9,

- ```
kCLOCK_MiscDivBy64 = 63 }
```

*Clock divider value.*
- enum `clock_usb_src_t` {
 

```
kCLOCK_Usb480M = 0,
```

```
kCLOCK_UsbSrcUnused = (int)0xFFFFFFFFU }
```

*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_PliClkSrc24M = 0U,
```

```
kCLOCK_PlISrcClkPN = 1U }
```

*PLL clock source, bypass cloco source also.*
  - enum `clock_pll_t` {
 

```
kCLOCK_PlISys = CCM_ANALOG_TUPLE(PLL_SYS_OFFSET, CCM_ANALOG_PLL_SYS_ENABLE_SHIFT),
```

```
kCLOCK_PlIUsb1 = CCM_ANALOG_TUPLE(PLL_USB1_OFFSET, CCM_ANALOG_PLL_USB1_ENABLE_SHIFT),
```

```
kCLOCK_PlIAudio = CCM_ANALOG_TUPLE(PLL_AUDIO_OFFSET, CCM_ANALOG_PLL_AUDIO_ENABLE_SHIFT),
```

```
kCLOCK_PlIEnet500M = CCM_ANALOG_TUPLE(PLL_ENET_OFFSET, CCM_ANALOG_PLL_ENET_500M_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_OutputPliUsb1Sw = 0U,
```

```
kCLOCK_OutputPliSys = 1U,
```

```
kCLOCK_OutputPliENET = 2U,
```

```
kCLOCK_OutputCoreClk = 0xBU,
```

```
kCLOCK_OutputIpgClk = 0xCU,
```

```
kCLOCK_OutputPerClk = 0xDU,
```

```
kCLOCK_OutputPli4MainClk = 0xFU,
```

```
kCLOCK_DisableClockOutput1 = 0x10U }
```

*The enumerater of clock output1's clock source, such as USB1 PLL, SYS PLL and so on.*
  - enum `clock_output2_selection_t` {

```
kCLOCK_OutputLpi2cClk = 6U,
kCLOCK_OutputOscClk = 0xEU,
kCLOCK_OutputLpspiClk = 0x10U,
kCLOCK_OutputSai1Clk = 0x12U,
kCLOCK_OutputSai3Clk = 0x14U,
kCLOCK_OutputTraceClk = 0x16U,
kCLOCK_OutputFlexspiClk = 0x1BU,
kCLOCK_OutputUartClk = 0x1CU,
kCLOCK_OutputSpdif0Clk = 0x1DU,
kCLOCK_DisableClockOutput2 = 0x1FU }
```

*The enumerater of clock output2's clock source, such as USDHCI 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\_FlexspiClkRoot = 0U,
   
kCLOCK\_LpspiClkRoot,
   
kCLOCK\_TraceClkRoot,
   
kCLOCK\_Sai1ClkRoot,
   
kCLOCK\_Sai3ClkRoot,
   
kCLOCK\_Lpi2cClkRoot,
   
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\_GetCoreFreq** (void)  
*Gets the CORE 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 **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)
- #define **SDK\_DEVICE\_MAXIMUM\_CPU\_CLOCK\_FREQUENCY** (500000000UL)

## 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)

- static void **CLOCK\_SetRtcXtalFreq** (uint32\_t freq)
 

*Set the XTAL (24M OSC) frequency based on board setting.*
- void **CLOCK\_InitRcOsc24M** (void)
 

*Set the RTC XTAL (32K OSC) frequency based on board setting.*
- void **CLOCK\_DeinitRcOsc24M** (void)
 

*Initialize the RC oscillator 24MHz clock.*
- 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 \
}
```

### 4.3.4 #define AOI\_CLOCKS

**Value:**

```
{
 kCLOCK_Aoi \
}
```

#### 4.3.5 #define DCDC\_CLOCKS

**Value:**

```
{
 kCLOCK_Dcdc \
}
```

#### 4.3.6 #define DCP\_CLOCKS

**Value:**

```
{
 kCLOCK_Dcp \
}
```

#### 4.3.7 #define DMAMUX\_CLOCKS

**Value:**

```
{
 kCLOCK_Dma \
}
```

#### 4.3.8 #define EDMA\_CLOCKS

**Value:**

```
{
 kCLOCK_Dma \
}
```

#### 4.3.9 #define EWM\_CLOCKS

**Value:**

```
{
 kCLOCK_Ewm0 \
}
```

### 4.3.10 #define FLEXIO\_CLOCKS

**Value:**

```
{
 kCLOCK_Invalid, \
 kCLOCK_Flexio1 \
}
```

### 4.3.11 #define FEXRAM\_CLOCKS

**Value:**

```
{
 kCLOCK_FlexRam \
}
```

### 4.3.12 #define FLEXSPI\_CLOCKS

**Value:**

```
{
 kCLOCK_FlexSpi \
}
```

### 4.3.13 #define GPIO\_CLOCKS

**Value:**

```
{
 kCLOCK_Invalid, kCLOCK_Gpio1, kCLOCK_Gpio2, kCLOCK_Invalid,
 kCLOCK_Invalid, kCLOCK_Gpio5 \
}
```

### 4.3.14 #define GPT\_CLOCKS

**Value:**

```
{
 kCLOCK_Invalid, \
 kCLOCK_Gpt1, kCLOCK_Gpt2 \
}
```

#### 4.3.15 #define KPP\_CLOCKS

**Value:**

```
{
 kCLOCK_Kpp \
}
```

#### 4.3.16 #define LPI2C\_CLOCKS

**Value:**

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

#### 4.3.17 #define LPSPI\_CLOCKS

**Value:**

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

#### 4.3.18 #define LPUART\_CLOCKS

**Value:**

```
{
 kCLOCK_IpInvalid, kCLOCK_Lpuart1, kCLOCK_Lpuart2,
 kCLOCK_Lpuart3, kCLOCK_Lpuart4 \
}
```

#### 4.3.19 #define OCRAM\_EXSC\_CLOCKS

**Value:**

```
{
 kCLOCK_OcramExsc \
}
```

#### 4.3.20 #define PIT\_CLOCKS

**Value:**

```
{
 kCLOCK_Pit \
}
```

#### 4.3.21 #define PWM\_CLOCKS

**Value:**

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

#### 4.3.22 #define RTWDOG\_CLOCKS

**Value:**

```
{
 kCLOCK_Wdog3 \
}
```

#### 4.3.23 #define SAI\_CLOCKS

**Value:**

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

#### 4.3.24 #define TRNG\_CLOCKS

**Value:**

```
{
 kCLOCK_Trng \
}
```

**4.3.25 #define WDOG\_CLOCKS****Value:**

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

**4.3.26 #define SPDIF\_CLOCKS****Value:**

```
{
 kCLOCK_Spdif \
}
```

**4.3.27 #define XBARA\_CLOCKS****Value:**

```
{
 kCLOCK_Xbar1 \
}
```

**4.3.28 #define kCLOCK\_CoreSysClk kCLOCK\_CpuClk****4.3.29 #define CLOCK\_GetCoreSysClkFreq CLOCK\_GetCpuClkFreq****4.4 Enumeration Type Documentation****4.4.1 enum clock\_name\_t**

Enumerator

*kCLOCK\_CpuClk* CPU clock.*kCLOCK\_CoreClk* CORE clock.*kCLOCK\_IpgClk* IPG clock.*kCLOCK\_PerClk* PER clock.*kCLOCK\_OscClk* OSC clock selected by PMU\_LOWPWR\_CTRL[OSC\_SEL].*kCLOCK\_RtcClk* RTC clock. (RTCKLK)*kCLOCK\_Usb1PllClk* USB1PLLCLK.*kCLOCK\_Usb1PllPfd0Clk* USB1PLLPDF0CLK.

*kCLOCK\_Usb1PllPfd1Clk* USB1PLLPFD1CLK.  
*kCLOCK\_Usb1PllPfd2Clk* USB1PLLPFD2CLK.  
*kCLOCK\_Usb1PllPfd3Clk* USB1PLLPFD3CLK.  
*kCLOCK\_Usb1SwClk* USB1PLLSWCLK.  
*kCLOCK\_Usb1Sw60MClk* USB1PLLSw60MCLK.  
*kCLOCK\_Usb1Sw80MClk* USB1PLLSw80MCLK.  
*kCLOCK\_SysPllClk* SYSPLLCLK.  
*kCLOCK\_SysPllPfd0Clk* SYSPLLPDF0CLK.  
*kCLOCK\_SysPllPfd1Clk* SYSPLLPDF1CLK.  
*kCLOCK\_SysPllPfd2Clk* SYSPLLPDF2CLK.  
*kCLOCK\_SysPllPfd3Clk* SYSPLLPDF3CLK.  
*kCLOCK\_EnetPll500MClk* Enet PLLCLK ref\_enetpll500M.  
*kCLOCK\_AudioPllClk* Audio PLLCLK.  
*kCLOCK\_PeriphClk2* Periph CLK2 selection.  
*kCLOCK\_FlexspiSel* Flexspi selection.  
*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\_FlexSpiExsc* CCGR0, CG3.  
*kCLOCK\_Sim\_m\_clk\_r* CCGR0, CG4.  
*kCLOCK\_Dcp* CCGR0, CG5.  
*kCLOCK\_Lpuart3* CCGR0, CG6.  
*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\_Pit* CCGR1, CG6.  
*kCLOCK\_Adc1* CCGR1, CG8.  
*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\_Ocotp* CCGR2, CG6.  
*kCLOCK\_Xbar1* CCGR2, CG11.  
*kCLOCK\_Aoi* CCGR3, CG4.  
*kCLOCK\_Ewm0* CCGR3, CG7.  
*kCLOCK\_Wdog1* CCGR3, CG8.  
*kCLOCK\_FlexRam* CCGR3, CG9.  
*kCLOCK\_IomuxcSnvsGpr* CCGR3, CG15.  
*kCLOCK\_Sim\_m7\_clk\_r* CCGR4, CG0.  
*kCLOCK\_Iomuxc* CCGR4, CG1.  
*kCLOCK\_IomuxcGpr* CCGR4, CG2.  
*kCLOCK\_SimM7* CCGR4, CG4.  
*kCLOCK\_SimM* CCGR4, CG6.  
*kCLOCK\_SimEms* CCGR4, CG7.  
*kCLOCK\_Pwm1* CCGR4, CG8.  
*kCLOCK\_Dma\_ps* CCGR4, CG15..  
*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\_Spdif* CCGR5, CG7.  
*kCLOCK\_Sai1* CCGR5, CG9.  
*kCLOCK\_Sai3* CCGR5, CG11.  
*kCLOCK\_Lpuart1* CCGR5, CG12.  
*kCLOCK\_SnvsHp* CCGR5, CG14.  
*kCLOCK\_SnvsLp* CCGR5, CG15.  
*kCLOCK\_UsbOh3* CCGR6, CG0.  
*kCLOCK\_Dcdc* CCGR6, CG3.  
*kCLOCK\_FlexSpi* CCGR6, CG5.  
*kCLOCK\_Trng* CCGR6, CG6.  
*kCLOCK\_SimPer* CCGR6, CG10.  
*kCLOCK\_Anadig* CCGR6, CG11.

#### 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\_Pl3SwMux* pll3\_sw\_clk mux name  
*kCLOCK\_PeriphMux* periph 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\_FlexspiSrcMux* flexspi SRC mux name  
*kCLOCK\_Sai3Mux* sai3 mux name  
*kCLOCK\_Sai1Mux* sai1 mux name  
*kCLOCK\_PerclkMux* perclk mux name  
*kCLOCK\_Flexio1Mux* flexio1 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_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_AdcDiv adc name
kCLOCK_TraceDiv trace 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_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_AhbDivBy1 Ahb clock divider set to divided by 1.
kCLOCK_AhbDivBy2 Ahb clock divider set to divided by 2.
kCLOCK_AhbDivBy3 Ahb clock divider set to divided by 3.
kCLOCK_AhbDivBy4 Ahb clock divider set to divided by 4.
kCLOCK_AhbDivBy5 Ahb clock divider set to divided by 5.
kCLOCK_AhbDivBy6 Ahb clock divider set to divided by 6.
kCLOCK_AhbDivBy7 Ahb clock divider set to divided by 7.
kCLOCK_AhbDivBy8 Ahb clock divider set to divided by 8.
kCLOCK_IpgDivBy1 ipg clock divider set to divided by 1.
kCLOCK_IpgDivBy2 ipg clock divider set to divided by 2.
kCLOCK_IpgDivBy3 ipg clock divider set to divided by 3.
kCLOCK_IpgDivBy4 ipg clock divider set to divided by 4.
```

|                             |                                            |
|-----------------------------|--------------------------------------------|
| <b>kCLOCK_LpspiDivBy1</b>   | lpspi clock divider set to divided by 1.   |
| <b>kCLOCK_LpspiDivBy2</b>   | lpspi clock divider set to divided by 2.   |
| <b>kCLOCK_LpspiDivBy3</b>   | lpspi clock divider set to divided by 3.   |
| <b>kCLOCK_LpspiDivBy4</b>   | lpspi clock divider set to divided by 4.   |
| <b>kCLOCK_LpspiDivBy5</b>   | lpspi clock divider set to divided by 5.   |
| <b>kCLOCK_LpspiDivBy6</b>   | lpspi clock divider set to divided by 6.   |
| <b>kCLOCK_LpspiDivBy7</b>   | lpspi clock divider set to divided by 7.   |
| <b>kCLOCK_LpspiDivBy8</b>   | lpspi clock divider set to divided by 8.   |
| <b>kCLOCK_LpspiDivBy9</b>   | lpspi clock divider set to divided by 9.   |
| <b>kCLOCK_LpspiDivBy10</b>  | lpspi clock divider set to divided by 10.  |
| <b>kCLOCK_LpspiDivBy11</b>  | lpspi clock divider set to divided by 11.  |
| <b>kCLOCK_LpspiDivBy12</b>  | lpspi clock divider set to divided by 12.  |
| <b>kCLOCK_LpspiDivBy13</b>  | lpspi clock divider set to divided by 13.  |
| <b>kCLOCK_LpspiDivBy14</b>  | lpspi clock divider set to divided by 14.  |
| <b>kCLOCK_LpspiDivBy15</b>  | lpspi clock divider set to divided by 15.  |
| <b>kCLOCK_LpspiDivBy16</b>  | lpspi clock divider set to divided by 16.  |
| <b>kCLOCK_FlexspiDivBy1</b> | flexspi clock divider set to divided by 1. |
| <b>kCLOCK_FlexspiDivBy2</b> | flexspi clock divider set to divided by 2. |
| <b>kCLOCK_FlexspiDivBy3</b> | flexspi clock divider set to divided by 3. |
| <b>kCLOCK_FlexspiDivBy4</b> | flexspi clock divider set to divided by 4. |
| <b>kCLOCK_FlexspiDivBy5</b> | flexspi clock divider set to divided by 5. |
| <b>kCLOCK_FlexspiDivBy6</b> | flexspi clock divider set to divided by 6. |
| <b>kCLOCK_FlexspiDivBy7</b> | flexspi clock divider set to divided by 7. |
| <b>kCLOCK_FlexspiDivBy8</b> | flexspi clock divider set to divided by 8. |
| <b>kCLOCK_AdcDivBy8</b>     | adc clock divider set to divided by 8.     |
| <b>kCLOCK_AdcDivBy12</b>    | adc clock divider set to divided by 12.    |
| <b>kCLOCK_AdcDivBy16</b>    | adc clock divider set to divided by 16.    |
| <b>kCLOCK_TraceDivBy1</b>   | trace clock divider set to divided by 1.   |
| <b>kCLOCK_TraceDivBy2</b>   | trace clock divider set to divided by 2.   |
| <b>kCLOCK_TraceDivBy3</b>   | trace clock divider set to divided by 3.   |
| <b>kCLOCK_TraceDivBy4</b>   | trace clock divider set to divided by 4.   |
| <b>kCLOCK_TraceDivBy5</b>   | trace clock divider set to divided by 5.   |
| <b>kCLOCK_TraceDivBy6</b>   | trace clock divider set to divided by 6.   |
| <b>kCLOCK_TraceDivBy7</b>   | trace clock divider set to divided by 7.   |
| <b>kCLOCK_TraceDivBy8</b>   | trace clock divider set to divided by 8.   |
| <b>kCLOCK_TraceDivBy9</b>   | trace clock divider set to divided by 9.   |
| <b>kCLOCK_TraceDivBy10</b>  | trace clock divider set to divided by 10.  |
| <b>kCLOCK_TraceDivBy11</b>  | trace clock divider set to divided by 11.  |
| <b>kCLOCK_TraceDivBy12</b>  | trace clock divider set to divided by 12.  |
| <b>kCLOCK_TraceDivBy13</b>  | trace clock divider set to divided by 13.  |
| <b>kCLOCK_TraceDivBy14</b>  | trace clock divider set to divided by 14.  |
| <b>kCLOCK_TraceDivBy15</b>  | trace clock divider set to divided by 15.  |
| <b>kCLOCK_TraceDivBy16</b>  | trace clock divider set to divided by 16.  |
| <b>kCLOCK_Flexio1DivBy1</b> | flexio1 clock divider set to divided by 1. |
| <b>kCLOCK_Flexio1DivBy2</b> | flexio1 clock divider set to divided by 2. |

|                                |                                                |
|--------------------------------|------------------------------------------------|
| <b>kCLOCK_Flexio1DivBy3</b>    | flexio1 clock divider set to divided by 3.     |
| <b>kCLOCK_Flexio1DivBy4</b>    | flexio1 clock divider set to divided by 4.     |
| <b>kCLOCK_Flexio1DivBy5</b>    | flexio1 clock divider set to divided by 5.     |
| <b>kCLOCK_Flexio1DivBy6</b>    | flexio1 clock divider set to divided by 6.     |
| <b>kCLOCK_Flexio1DivBy7</b>    | flexio1 clock divider set to divided by 7.     |
| <b>kCLOCK_Flexio1DivBy8</b>    | flexio1 clock divider set to divided by 8.     |
| <b>kCLOCK_Flexio1DivBy9</b>    | flexio1 clock divider set to divided by 9.     |
| <b>kCLOCK_Flexio1DivBy10</b>   | flexio1 clock divider set to divided by 10.    |
| <b>kCLOCK_Flexio1DivBy11</b>   | flexio1 clock divider set to divided by 11.    |
| <b>kCLOCK_Flexio1DivBy12</b>   | flexio1 clock divider set to divided by 12.    |
| <b>kCLOCK_Flexio1DivBy13</b>   | flexio1 clock divider set to divided by 13.    |
| <b>kCLOCK_Flexio1DivBy14</b>   | flexio1 clock divider set to divided by 14.    |
| <b>kCLOCK_Flexio1DivBy15</b>   | flexio1 clock divider set to divided by 15.    |
| <b>kCLOCK_Flexio1DivBy16</b>   | flexio1 clock divider set to divided by 16.    |
| <b>kCLOCK_Sai3PreDivBy1</b>    | sai3 pre clock divider set to divided by 1.    |
| <b>kCLOCK_Sai3PreDivBy2</b>    | sai3 pre clock divider set to divided by 2.    |
| <b>kCLOCK_Sai3PreDivBy3</b>    | sai3 pre clock divider set to divided by 3.    |
| <b>kCLOCK_Sai3PreDivBy4</b>    | sai3 pre clock divider set to divided by 4.    |
| <b>kCLOCK_Sai3PreDivBy5</b>    | sai3 pre clock divider set to divided by 5.    |
| <b>kCLOCK_Sai3PreDivBy6</b>    | sai3 pre clock divider set to divided by 6.    |
| <b>kCLOCK_Sai3PreDivBy7</b>    | sai3 pre clock divider set to divided by 7.    |
| <b>kCLOCK_Sai3PreDivBy8</b>    | sai3 pre clock divider set to divided by 8.    |
| <b>kCLOCK_Flexio1PreDivBy1</b> | flexio1 pre clock divider set to divided by 1. |
| <b>kCLOCK_Flexio1PreDivBy2</b> | flexio1 pre clock divider set to divided by 2. |
| <b>kCLOCK_Flexio1PreDivBy3</b> | flexio1 pre clock divider set to divided by 3. |
| <b>kCLOCK_Flexio1PreDivBy4</b> | flexio1 pre clock divider set to divided by 4. |
| <b>kCLOCK_Flexio1PreDivBy5</b> | flexio1 pre clock divider set to divided by 5. |
| <b>kCLOCK_Flexio1PreDivBy6</b> | flexio1 pre clock divider set to divided by 6. |
| <b>kCLOCK_Flexio1PreDivBy7</b> | flexio1 pre clock divider set to divided by 7. |
| <b>kCLOCK_Flexio1PreDivBy8</b> | flexio1 pre clock divider set to divided by 8. |
| <b>kCLOCK_Sai1PreDivBy1</b>    | sai1 pre clock divider set to divided by 1.    |
| <b>kCLOCK_Sai1PreDivBy2</b>    | sai1 pre clock divider set to divided by 2.    |
| <b>kCLOCK_Sai1PreDivBy3</b>    | sai1 pre clock divider set to divided by 3.    |
| <b>kCLOCK_Sai1PreDivBy4</b>    | sai1 pre clock divider set to divided by 4.    |
| <b>kCLOCK_Sai1PreDivBy5</b>    | sai1 pre clock divider set to divided by 5.    |
| <b>kCLOCK_Sai1PreDivBy6</b>    | sai1 pre clock divider set to divided by 6.    |
| <b>kCLOCK_Sai1PreDivBy7</b>    | sai1 pre clock divider set to divided by 7.    |
| <b>kCLOCK_Sai1PreDivBy8</b>    | sai1 pre clock divider set to divided by 8.    |
| <b>kCLOCK_Spdif0PreDivBy1</b>  | spdif pre clock divider set to divided by 1.   |
| <b>kCLOCK_Spdif0PreDivBy2</b>  | spdif pre clock divider set to divided by 2.   |
| <b>kCLOCK_Spdif0PreDivBy3</b>  | spdif pre clock divider set to divided by 3.   |
| <b>kCLOCK_Spdif0PreDivBy4</b>  | spdif pre clock divider set to divided by 4.   |
| <b>kCLOCK_Spdif0PreDivBy5</b>  | spdif pre clock divider set to divided by 5.   |
| <b>kCLOCK_Spdif0PreDivBy6</b>  | spdif pre clock divider set to divided by 6.   |
| <b>kCLOCK_Spdif0PreDivBy7</b>  | spdif pre clock divider set to divided by 7.   |

|                               |                                               |
|-------------------------------|-----------------------------------------------|
| <b>kCLOCK_Spdif0PreDivBy8</b> | spdif pre clock divider set to divided by 8.  |
| <b>kCLOCK_Spdif0DivBy1</b>    | spdif clock divider set to divided by 1.      |
| <b>kCLOCK_Spdif0DivBy2</b>    | spdif clock divider set to divided by 2.      |
| <b>kCLOCK_Spdif0DivBy3</b>    | spdif clock divider set to divided by 3.      |
| <b>kCLOCK_Spdif0DivBy4</b>    | spdif clock divider set to divided by 4.      |
| <b>kCLOCK_Spdif0DivBy5</b>    | spdif clock divider set to divided by 5.      |
| <b>kCLOCK_Spdif0DivBy6</b>    | spdif clock divider set to divided by 6.      |
| <b>kCLOCK_Spdif0DivBy7</b>    | spdif clock divider set to divided by 7.      |
| <b>kCLOCK_Spdif0DivBy8</b>    | spdif clock divider set to divided by 8.      |
| <b>kCLOCK_MiscDivBy1</b>      | Misc divider like LPI2C set to divided by 1.  |
| <b>kCLOCK_MiscDivBy2</b>      | Misc divider like LPI2C set to divided by 2.  |
| <b>kCLOCK_MiscDivBy3</b>      | Misc divider like LPI2C set to divided by 3.  |
| <b>kCLOCK_MiscDivBy4</b>      | Misc divider like LPI2C set to divided by 4.  |
| <b>kCLOCK_MiscDivBy5</b>      | Misc divider like LPI2C set to divided by 5.  |
| <b>kCLOCK_MiscDivBy6</b>      | Misc divider like LPI2C set to divided by 6.  |
| <b>kCLOCK_MiscDivBy7</b>      | Misc divider like LPI2C set to divided by 7.  |
| <b>kCLOCK_MiscDivBy8</b>      | Misc divider like LPI2C set to divided by 8.  |
| <b>kCLOCK_MiscDivBy9</b>      | Misc divider like LPI2C set to divided by 9.  |
| <b>kCLOCK_MiscDivBy10</b>     | Misc divider like LPI2C set to divided by 10. |
| <b>kCLOCK_MiscDivBy11</b>     | Misc divider like LPI2C set to divided by 11. |
| <b>kCLOCK_MiscDivBy12</b>     | Misc divider like LPI2C set to divided by 12. |
| <b>kCLOCK_MiscDivBy13</b>     | Misc divider like LPI2C set to divided by 13. |
| <b>kCLOCK_MiscDivBy14</b>     | Misc divider like LPI2C set to divided by 14. |
| <b>kCLOCK_MiscDivBy15</b>     | Misc divider like LPI2C set to divided by 15. |
| <b>kCLOCK_MiscDivBy16</b>     | Misc divider like LPI2C set to divided by 16. |
| <b>kCLOCK_MiscDivBy17</b>     | Misc divider like LPI2C set to divided by 17. |
| <b>kCLOCK_MiscDivBy18</b>     | Misc divider like LPI2C set to divided by 18. |
| <b>kCLOCK_MiscDivBy19</b>     | Misc divider like LPI2C set to divided by 19. |
| <b>kCLOCK_MiscDivBy20</b>     | Misc divider like LPI2C set to divided by 20. |
| <b>kCLOCK_MiscDivBy21</b>     | Misc divider like LPI2C set to divided by 21. |
| <b>kCLOCK_MiscDivBy22</b>     | Misc divider like LPI2C set to divided by 22. |
| <b>kCLOCK_MiscDivBy23</b>     | Misc divider like LPI2C set to divided by 23. |
| <b>kCLOCK_MiscDivBy24</b>     | Misc divider like LPI2C set to divided by 24. |
| <b>kCLOCK_MiscDivBy25</b>     | Misc divider like LPI2C set to divided by 25. |
| <b>kCLOCK_MiscDivBy26</b>     | Misc divider like LPI2C set to divided by 26. |
| <b>kCLOCK_MiscDivBy27</b>     | Misc divider like LPI2C set to divided by 27. |
| <b>kCLOCK_MiscDivBy28</b>     | Misc divider like LPI2C set to divided by 28. |
| <b>kCLOCK_MiscDivBy29</b>     | Misc divider like LPI2C set to divided by 29. |
| <b>kCLOCK_MiscDivBy30</b>     | Misc divider like LPI2C set to divided by 30. |
| <b>kCLOCK_MiscDivBy31</b>     | Misc divider like LPI2C set to divided by 31. |
| <b>kCLOCK_MiscDivBy32</b>     | Misc divider like LPI2C set to divided by 32. |
| <b>kCLOCK_MiscDivBy33</b>     | Misc divider like LPI2C set to divided by 33. |
| <b>kCLOCK_MiscDivBy34</b>     | Misc divider like LPI2C set to divided by 34. |
| <b>kCLOCK_MiscDivBy35</b>     | Misc divider like LPI2C set to divided by 35. |
| <b>kCLOCK_MiscDivBy36</b>     | Misc divider like LPI2C set to divided by 36. |

|                           |                                               |
|---------------------------|-----------------------------------------------|
| <i>kCLOCK_MiscDivBy37</i> | Misc divider like LPI2C set to divided by 37. |
| <i>kCLOCK_MiscDivBy38</i> | Misc divider like LPI2C set to divided by 38. |
| <i>kCLOCK_MiscDivBy39</i> | Misc divider like LPI2C set to divided by 39. |
| <i>kCLOCK_MiscDivBy40</i> | Misc divider like LPI2C set to divided by 40. |
| <i>kCLOCK_MiscDivBy41</i> | Misc divider like LPI2C set to divided by 41. |
| <i>kCLOCK_MiscDivBy42</i> | Misc divider like LPI2C set to divided by 42. |
| <i>kCLOCK_MiscDivBy43</i> | Misc divider like LPI2C set to divided by 43. |
| <i>kCLOCK_MiscDivBy44</i> | Misc divider like LPI2C set to divided by 44. |
| <i>kCLOCK_MiscDivBy45</i> | Misc divider like LPI2C set to divided by 45. |
| <i>kCLOCK_MiscDivBy46</i> | Misc divider like LPI2C set to divided by 46. |
| <i>kCLOCK_MiscDivBy47</i> | Misc divider like LPI2C set to divided by 47. |
| <i>kCLOCK_MiscDivBy48</i> | Misc divider like LPI2C set to divided by 48. |
| <i>kCLOCK_MiscDivBy49</i> | Misc divider like LPI2C set to divided by 49. |
| <i>kCLOCK_MiscDivBy50</i> | Misc divider like LPI2C set to divided by 50. |
| <i>kCLOCK_MiscDivBy51</i> | Misc divider like LPI2C set to divided by 51. |
| <i>kCLOCK_MiscDivBy52</i> | Misc divider like LPI2C set to divided by 52. |
| <i>kCLOCK_MiscDivBy53</i> | Misc divider like LPI2C set to divided by 53. |
| <i>kCLOCK_MiscDivBy54</i> | Misc divider like LPI2C set to divided by 54. |
| <i>kCLOCK_MiscDivBy55</i> | Misc divider like LPI2C set to divided by 55. |
| <i>kCLOCK_MiscDivBy56</i> | Misc divider like LPI2C set to divided by 56. |
| <i>kCLOCK_MiscDivBy57</i> | Misc divider like LPI2C set to divided by 57. |
| <i>kCLOCK_MiscDivBy58</i> | Misc divider like LPI2C set to divided by 58. |
| <i>kCLOCK_MiscDivBy59</i> | Misc divider like LPI2C set to divided by 59. |
| <i>kCLOCK_MiscDivBy60</i> | Misc divider like LPI2C set to divided by 60. |
| <i>kCLOCK_MiscDivBy61</i> | Misc divider like LPI2C set to divided by 61. |
| <i>kCLOCK_MiscDivBy62</i> | Misc divider like LPI2C set to divided by 62. |
| <i>kCLOCK_MiscDivBy63</i> | Misc divider like LPI2C set to divided by 63. |
| <i>kCLOCK_MiscDivBy64</i> | 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\_PllEnet500M* PLL ENET.

#### 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\_OutputPllENET* Selects ENET PLL clock(Divided by 2) output.

*kCLOCK\_OutputCoreClk* Selects Core 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 CLKO1.

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

Enumerator

- kCLOCK\_OutputLpi2cClk* Selects LPI2C clock root output.
- kCLOCK\_OutputOscClk* Selects OSC output.
- kCLOCK\_OutputLpspiClk* Selects LPSPI clock root output.
- kCLOCK\_OutputSai1Clk* Selects SAI1 clock root output.
- kCLOCK\_OutputSai3Clk* Selects SAI3 clock root output.
- kCLOCK\_OutputTraceClk* Selects Trace 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 CLKO2.

#### 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\_FlexspiClkRoot* FLEXSPI clock root.
- kCLOCK\_LpspiClkRoot* LPSPI clock root.
- kCLOCK\_TraceClkRoot* Trace clock root.
- kCLOCK\_Sai1ClkRoot* SAI1 clock root.
- kCLOCK\_Sai3ClkRoot* SAI3 clock root.
- kCLOCK\_Lpi2cClkRoot* LPI2C 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\\_UartDiv](#), [kCLOCK\\_Sai3Div](#), [kCLOCK\\_Sai1Div](#), [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\_GetCoreFreq( void )**

Returns

The CORE clock frequency value in hertz.

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

Returns

The IPG clock frequency value in hertz.

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

Returns

The PER clock frequency value in hertz.

**4.5.13 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

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

Returns

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

**4.5.15 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.

**4.5.16 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.

**4.5.17 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.

**4.5.18 void CLOCK\_SwitchOsc( clock\_osc\_t *osc* )**

This function switches the OSC source for SoC.

Parameters

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

#### 4.5.19 static uint32\_t CLOCK\_GetRtcFreq( void ) [inline], [static]

Returns

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

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

Parameters

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

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

Parameters

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

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

This function only enables the access to USB HS prepheral, 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 <a href="#">kCLOCK_UsbSrcUnused</a> . |
| <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(240000000);
*
```

### 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

## IOMUXC: IOMUX Controller

### 5.1 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, 4))`  
*IOMUXC driver version 2.0.3.*

### Pin function ID

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

- `#define IOMUXC_GPIO_AD_14_LPI2C1_SCL 0x401F8010U, 0x0U, 0x401F81C0U, 0x0U, 0x401F80C0U`
- `#define IOMUXC_GPIO_AD_14_LPUART3_CTS_B 0x401F8010U, 0x1U, 0, 0, 0x401F80C0U`
- `#define IOMUXC_GPIO_AD_14_KPP_COL00 0x401F8010U, 0x2U, 0x401F819CU, 0x0U, 0x401F80C0U`
- `#define IOMUXC_GPIO_AD_14_LPUART4_CTS_B 0x401F8010U, 0x3U, 0, 0, 0x401F80C0U`
- `#define IOMUXC_GPIO_AD_14_FLEXIO1_IO26 0x401F8010U, 0x4U, 0, 0, 0x401F80C0U`
- `#define IOMUXC_GPIO_AD_14_GPIOMUX_IO28 0x401F8010U, 0x5U, 0, 0, 0x401F80C0U`
- `#define IOMUXC_GPIO_AD_14_REF_CLK_24M 0x401F8010U, 0x6U, 0, 0, 0x401F80C0U`
- `#define IOMUXC_GPIO_AD_14_XBAR1_INOUT02 0x401F8010U, 0x7U, 0, 0, 0x401F80C0U`
- `#define IOMUXC_GPIO_AD_13_LPI2C1_SDA 0x401F8014U, 0x0U, 0x401F81C4U, 0x0U, 0x401F80C4U`
- `#define IOMUXC_GPIO_AD_13_LPUART3_RTS_B 0x401F8014U, 0x1U, 0, 0, 0x401F80C4U`
- `#define IOMUXC_GPIO_AD_13_KPP_ROW00 0x401F8014U, 0x2U, 0x401F81ACU, 0x0U, 0x401F80C4U`
- `#define IOMUXC_GPIO_AD_13_LPUART4_RTS_B 0x401F8014U, 0x3U, 0, 0, 0x401F80C4U`
- `#define IOMUXC_GPIO_AD_13_FLEXIO1_IO25 0x401F8014U, 0x4U, 0, 0, 0x401F80C4U`
- `#define IOMUXC_GPIO_AD_13_GPIOMUX_IO27 0x401F8014U, 0x5U, 0, 0, 0x401F80C4U`
- `#define IOMUXC_GPIO_AD_13_ARM_NMI 0x401F8014U, 0x6U, 0x401F8210U, 0x0U, 0x401F80C4U`
- `#define IOMUXC_GPIO_AD_13_JTAG_TMS 0x401F8014U, 0x7U, 0, 0, 0x401F80C4U`
- `#define IOMUXC_GPIO_AD_12_LPSPi2_SCK 0x401F8018U, 0x0U, 0x401F81E4U, 0x0U, 0x401F80C8U`
- `#define IOMUXC_GPIO_AD_12_FLEXPWM1_PWM0_X 0x401F8018U, 0x1U, 0, 0, 0x401F80C8U`
- `#define IOMUXC_GPIO_AD_12_KPP_COL01 0x401F8018U, 0x2U, 0x401F81A0U, 0x0U, 0x401F80C8U`

- #define **IOMUXC\_GPIO\_AD\_12\_PIT\_TRIGGER01** 0x401F8018U, 0x3U, 0, 0, 0x401F80C8U
- #define **IOMUXC\_GPIO\_AD\_12\_FLEXIO1\_IO24** 0x401F8018U, 0x4U, 0, 0, 0x401F80C8U
- #define **IOMUXC\_GPIO\_AD\_12\_GPIOMUX\_IO26** 0x401F8018U, 0x5U, 0, 0, 0x401F80C8U
- #define **IOMUXC\_GPIO\_AD\_12\_USB\_OTG1\_PWR** 0x401F8018U, 0x6U, 0, 0, 0x401F80C8U
- #define **IOMUXC\_GPIO\_AD\_12\_JTAG\_TCK** 0x401F8018U, 0x7U, 0, 0, 0x401F80C8U
- #define **IOMUXC\_GPIO\_AD\_11\_LPSPI2\_PCS0** 0x401F801CU, 0x0U, 0x401F81E0U, 0x0U, 0x401F80CCU
- #define **IOMUXC\_GPIO\_AD\_11\_FLEXPWM1\_PWM1\_X** 0x401F801CU, 0x1U, 0, 0, 0x401F80CCU
- #define **IOMUXC\_GPIO\_AD\_11\_KPP\_ROW01** 0x401F801CU, 0x2U, 0x401F81B0U, 0x0U, 0x401F80CCU
- #define **IOMUXC\_GPIO\_AD\_11\_PIT\_TRIGGER02** 0x401F801CU, 0x3U, 0, 0, 0x401F80CCU
- #define **IOMUXC\_GPIO\_AD\_11\_FLEXIO1\_IO23** 0x401F801CU, 0x4U, 0, 0, 0x401F80CCU
- #define **IOMUXC\_GPIO\_AD\_11\_GPIOMUX\_IO25** 0x401F801CU, 0x5U, 0, 0, 0x401F80CCU
- #define **IOMUXC\_GPIO\_AD\_11\_WDOG1\_B** 0x401F801CU, 0x6U, 0, 0, 0x401F80CCU
- #define **IOMUXC\_GPIO\_AD\_11\_JTAG\_MOD** 0x401F801CU, 0x7U, 0, 0, 0x401F80CCU
- #define **IOMUXC\_GPIO\_AD\_10\_LPSPI2\_SDO** 0x401F8020U, 0x0U, 0x401F81ECU, 0x0U, 0x401F80D0U
- #define **IOMUXC\_GPIO\_AD\_10\_FLEXPWM1\_PWM2\_X** 0x401F8020U, 0x1U, 0, 0, 0x401F80D0U
- #define **IOMUXC\_GPIO\_AD\_10\_KPP\_COL02** 0x401F8020U, 0x2U, 0x401F81A4U, 0x0U, 0x401F80D0U
- #define **IOMUXC\_GPIO\_AD\_10\_PIT\_TRIGGER03** 0x401F8020U, 0x3U, 0, 0, 0x401F80D0U
- #define **IOMUXC\_GPIO\_AD\_10\_FLEXIO1\_IO22** 0x401F8020U, 0x4U, 0, 0, 0x401F80D0U
- #define **IOMUXC\_GPIO\_AD\_10\_GPIOMUX\_IO24** 0x401F8020U, 0x5U, 0, 0, 0x401F80D0U
- #define **IOMUXC\_GPIO\_AD\_10\_USB\_OTG1\_ID** 0x401F8020U, 0x6U, 0x401F8170U, 0x0U, 0x401F80D0U
- #define **IOMUXC\_GPIO\_AD\_10\_JTAG\_TDI** 0x401F8020U, 0x7U, 0, 0, 0x401F80D0U
- #define **IOMUXC\_GPIO\_AD\_09\_LPSPI2\_SDI** 0x401F8024U, 0x0U, 0x401F81E8U, 0x0U, 0x401F80D4U
- #define **IOMUXC\_GPIO\_AD\_09\_FLEXPWM1\_PWM3\_X** 0x401F8024U, 0x1U, 0, 0, 0x401F80D4U
- #define **IOMUXC\_GPIO\_AD\_09\_KPP\_ROW02** 0x401F8024U, 0x2U, 0x401F81B4U, 0x0U, 0x401F80D4U
- #define **IOMUXC\_GPIO\_AD\_09\_ARM\_TRACE\_SWO** 0x401F8024U, 0x3U, 0, 0, 0x401F80D4U
- #define **IOMUXC\_GPIO\_AD\_09\_FLEXIO1\_IO21** 0x401F8024U, 0x4U, 0, 0, 0x401F80D4U
- #define **IOMUXC\_GPIO\_AD\_09\_GPIOMUX\_IO23** 0x401F8024U, 0x5U, 0, 0, 0x401F80D4U
- #define **IOMUXC\_GPIO\_AD\_09\_REF\_CLK\_32K** 0x401F8024U, 0x6U, 0, 0, 0x401F80D4U
- #define **IOMUXC\_GPIO\_AD\_09\_JTAG\_TDO** 0x401F8024U, 0x7U, 0, 0, 0x401F80D4U
- #define **IOMUXC\_GPIO\_AD\_08\_LPI2C2\_SCL** 0x401F8028U, 0x0U, 0x401F81C8U, 0x0U, 0x401F80D8U
- #define **IOMUXC\_GPIO\_AD\_08\_LPUART3\_TXD** 0x401F8028U, 0x1U, 0x401F8204U, 0x0U, 0x401F80D8U
- #define **IOMUXC\_GPIO\_AD\_08\_ARM\_CM7\_TXEV** 0x401F8028U, 0x2U, 0, 0, 0x401F80D8U
- #define **IOMUXC\_GPIO\_AD\_08\_LPUART2\_CTS\_B** 0x401F8028U, 0x3U, 0, 0, 0x401F80D8U
- #define **IOMUXC\_GPIO\_AD\_08\_GPT2\_COMPARE3** 0x401F8028U, 0x4U, 0, 0, 0x401F80D8U
- #define **IOMUXC\_GPIO\_AD\_08\_GPIOMUX\_IO22** 0x401F8028U, 0x5U, 0, 0, 0x401F80D8U
- #define **IOMUXC\_GPIO\_AD\_08\_EWM\_OUT\_B** 0x401F8028U, 0x6U, 0, 0, 0x401F80D8U
- #define **IOMUXC\_GPIO\_AD\_08\_JTAG\_TRSTB** 0x401F8028U, 0x7U, 0, 0, 0x401F80D8U

- #define **IOMUXC\_GPIO\_AD\_07\_LPI2C2\_SDA** 0x401F802CU, 0x0U, 0x401F81CCU, 0x0U, 0x401F80DCU
- #define **IOMUXC\_GPIO\_AD\_07\_LPUART3\_RXD** 0x401F802CU, 0x1U, 0x401F8200U, 0x0U, 0x401F80DCU
- #define **IOMUXC\_GPIO\_AD\_07\_ARM\_CM7\_RXEV** 0x401F802CU, 0x2U, 0x401F8220U, 0x0U, 0x401F80DCU
- #define **IOMUXC\_GPIO\_AD\_07\_LPUART2\_RTS\_B** 0x401F802CU, 0x3U, 0, 0, 0x401F80DCU
- #define **IOMUXC\_GPIO\_AD\_07\_GPT2\_CAPTURE2** 0x401F802CU, 0x4U, 0, 0, 0x401F80DCU
- #define **IOMUXC\_GPIO\_AD\_07\_GPIOMUX\_IO21** 0x401F802CU, 0x5U, 0, 0, 0x401F80DCU
- #define **IOMUXC\_GPIO\_AD\_07\_OCOTP\_FUSE\_LATCHED** 0x401F802CU, 0x6U, 0, 0, 0x401F80DCU
- #define **IOMUXC\_GPIO\_AD\_07\_XBAR1\_INOUT03** 0x401F802CU, 0x7U, 0, 0, 0x401F80DCU
- #define **IOMUXC\_GPIO\_AD\_06\_LPSP1\_SCK** 0x401F8030U, 0x0U, 0x401F81D4U, 0x0U, 0x401F80E0U
- #define **IOMUXC\_GPIO\_AD\_06\_PIT\_TRIGGER00** 0x401F8030U, 0x1U, 0, 0, 0x401F80E0U
- #define **IOMUXC\_GPIO\_AD\_06\_FLEXPWM1\_PWM3\_A** 0x401F8030U, 0x2U, 0x401F8180U, 0x0U, 0x401F80E0U
- #define **IOMUXC\_GPIO\_AD\_06\_KPP\_COL01** 0x401F8030U, 0x3U, 0x401F81A0U, 0x1U, 0x401F80E0U
- #define **IOMUXC\_GPIO\_AD\_06\_GPT2\_COMPARE2** 0x401F8030U, 0x4U, 0, 0, 0x401F80E0U
- #define **IOMUXC\_GPIO\_AD\_06\_GPIOMUX\_IO20** 0x401F8030U, 0x5U, 0, 0, 0x401F80E0U
- #define **IOMUXC\_GPIO\_AD\_06\_LPI2C1\_HREQ** 0x401F8030U, 0x6U, 0x401F81BCU, 0x0U, 0x401F80E0U
- #define **IOMUXC\_GPIO\_AD\_05\_LPSP1\_PCS0** 0x401F8034U, 0x0U, 0x401F81D0U, 0x0U, 0x401F80E4U
- #define **IOMUXC\_GPIO\_AD\_05\_PIT\_TRIGGER01** 0x401F8034U, 0x1U, 0, 0, 0x401F80E4U
- #define **IOMUXC\_GPIO\_AD\_05\_FLEXPWM1\_PWM3\_B** 0x401F8034U, 0x2U, 0x401F8190U, 0x0U, 0x401F80E4U
- #define **IOMUXC\_GPIO\_AD\_05\_KPP\_ROW01** 0x401F8034U, 0x3U, 0x401F81B0U, 0x1U, 0x401F80E4U
- #define **IOMUXC\_GPIO\_AD\_05\_GPT2\_CAPTURE1** 0x401F8034U, 0x4U, 0, 0, 0x401F80E4U
- #define **IOMUXC\_GPIO\_AD\_05\_GPIOMUX\_IO19** 0x401F8034U, 0x5U, 0, 0, 0x401F80E4U
- #define **IOMUXC\_GPIO\_AD\_04\_LPSP1\_SDO** 0x401F8038U, 0x0U, 0x401F81DCU, 0x0U, 0x401F80E8U
- #define **IOMUXC\_GPIO\_AD\_04\_PIT\_TRIGGER02** 0x401F8038U, 0x1U, 0, 0, 0x401F80E8U
- #define **IOMUXC\_GPIO\_AD\_04\_FLEXPWM1\_PWM2\_A** 0x401F8038U, 0x2U, 0x401F817CU, 0x0U, 0x401F80E8U
- #define **IOMUXC\_GPIO\_AD\_04\_KPP\_COL02** 0x401F8038U, 0x3U, 0x401F81A4U, 0x1U, 0x401F80E8U
- #define **IOMUXC\_GPIO\_AD\_04\_GPT2\_COMPARE1** 0x401F8038U, 0x4U, 0, 0, 0x401F80E8U
- #define **IOMUXC\_GPIO\_AD\_04\_GPIOMUX\_IO18** 0x401F8038U, 0x5U, 0, 0, 0x401F80E8U
- #define **IOMUXC\_GPIO\_AD\_04\_SNVS\_VIO\_5\_CTL** 0x401F8038U, 0x6U, 0, 0, 0x401F80E8U
- #define **IOMUXC\_GPIO\_AD\_03\_LPSP1\_SDI** 0x401F803CU, 0x0U, 0x401F81D8U, 0x0U, 0x401F80ECU
- #define **IOMUXC\_GPIO\_AD\_03\_PIT\_TRIGGER03** 0x401F803CU, 0x1U, 0, 0, 0x401F80ECU

- #define **IOMUXC\_GPIO\_AD\_03\_FLEXPWM1\_PWM2\_B** 0x401F803CU, 0x2U, 0x401F818CU, 0x0U, 0x401F80ECU
- #define **IOMUXC\_GPIO\_AD\_03\_KPP\_ROW02** 0x401F803CU, 0x3U, 0x401F81B4U, 0x1U, 0x401F80ECU
- #define **IOMUXC\_GPIO\_AD\_03\_GPT2\_CLK** 0x401F803CU, 0x4U, 0, 0, 0x401F80ECU
- #define **IOMUXC\_GPIO\_AD\_03\_GPIOMUX\_IO17** 0x401F803CU, 0x5U, 0, 0, 0x401F80ECU
- #define **IOMUXC\_GPIO\_AD\_03\_SNVS\_VIO\_5\_B** 0x401F803CU, 0x6U, 0, 0, 0x401F80ECU
- #define **IOMUXC\_GPIO\_AD\_03\_JTAG\_DE\_B** 0x401F803CU, 0x7U, 0, 0, 0x401F80ECU
- #define **IOMUXC\_GPIO\_AD\_02\_LPUART4\_TXD** 0x401F8040U, 0x0U, 0x401F820CU, 0x0U, 0x401F80F0U
- #define **IOMUXC\_GPIO\_AD\_02\_LPSP1\_PCS1** 0x401F8040U, 0x1U, 0, 0, 0x401F80F0U
- #define **IOMUXC\_GPIO\_AD\_02\_WDOG2\_B** 0x401F8040U, 0x2U, 0, 0, 0x401F80F0U
- #define **IOMUXC\_GPIO\_AD\_02\_LPI2C2\_SCL** 0x401F8040U, 0x3U, 0x401F81C8U, 0x1U, 0x401F80F0U
- #define **IOMUXC\_GPIO\_AD\_02\_MQS\_RIGHT** 0x401F8040U, 0x4U, 0, 0, 0x401F80F0U
- #define **IOMUXC\_GPIO\_AD\_02\_GPIOMUX\_IO16** 0x401F8040U, 0x5U, 0, 0, 0x401F80F0U
- #define **IOMUXC\_GPIO\_AD\_02\_ARM\_TRACE\_CLK** 0x401F8040U, 0x7U, 0, 0, 0x401F80F0U
- #define **IOMUXC\_GPIO\_AD\_01\_LPUART4\_RXD** 0x401F8044U, 0x0U, 0x401F8208U, 0x0U, 0x401F80F4U
- #define **IOMUXC\_GPIO\_AD\_01\_LPSP1\_PCS1** 0x401F8044U, 0x1U, 0, 0, 0x401F80F4U
- #define **IOMUXC\_GPIO\_AD\_01\_WDOG1\_ANY** 0x401F8044U, 0x2U, 0, 0, 0x401F80F4U
- #define **IOMUXC\_GPIO\_AD\_01\_LPI2C2\_SDA** 0x401F8044U, 0x3U, 0x401F81CCU, 0x1U, 0x401F80F4U
- #define **IOMUXC\_GPIO\_AD\_01\_MQS\_LEFT** 0x401F8044U, 0x4U, 0, 0, 0x401F80F4U
- #define **IOMUXC\_GPIO\_AD\_01\_GPIOMUX\_IO15** 0x401F8044U, 0x5U, 0, 0, 0x401F80F4U
- #define **IOMUXC\_GPIO\_AD\_01\_USB\_OTG1\_OC** 0x401F8044U, 0x6U, 0x401F821CU, 0x0U, 0x401F80F4U
- #define **IOMUXC\_GPIO\_AD\_01\_ARM\_TRACE\_SWO** 0x401F8044U, 0x7U, 0, 0, 0x401F80F4U
- #define **IOMUXC\_GPIO\_AD\_00\_LPUART2\_TXD** 0x401F8048U, 0x0U, 0x401F81FCU, 0x0U, 0x401F80F8U
- #define **IOMUXC\_GPIO\_AD\_00\_LPSP1\_PCS2** 0x401F8048U, 0x1U, 0, 0, 0x401F80F8U
- #define **IOMUXC\_GPIO\_AD\_00\_KPP\_COL03** 0x401F8048U, 0x2U, 0x401F81A8U, 0x0U, 0x401F80F8U
- #define **IOMUXC\_GPIO\_AD\_00\_USB\_OTG1\_PWR** 0x401F8048U, 0x3U, 0, 0, 0x401F80F8U
- #define **IOMUXC\_GPIO\_AD\_00\_FLEXIO1\_IO20** 0x401F8048U, 0x4U, 0, 0, 0x401F80F8U
- #define **IOMUXC\_GPIO\_AD\_00\_GPIOMUX\_IO14** 0x401F8048U, 0x5U, 0, 0, 0x401F80F8U
- #define **IOMUXC\_GPIO\_AD\_00\_ARM\_NMI** 0x401F8048U, 0x6U, 0x401F821OU, 0x1U, 0x401F80F8U
- #define **IOMUXC\_GPIO\_AD\_00\_ARM\_TRACE0** 0x401F8048U, 0x7U, 0, 0, 0x401F80F8U
- #define **IOMUXC\_GPIO\_SD\_14\_FLEXSPI\_A\_DQS** 0x401F804CU, 0x0U, 0x401F8194U, 0x0U, 0x401F80FCU
- #define **IOMUXC\_GPIO\_SD\_14\_FLEXSPI\_B\_DQS** 0x401F804CU, 0x1U, 0x401F8198U, 0x0U, 0x401F80FCU
- #define **IOMUXC\_GPIO\_SD\_13\_FLEXSPI\_B\_SCLK** 0x401F8050U, 0x0U, 0, 0, 0x401F8100U
- #define **IOMUXC\_GPIO\_SD\_13\_SAI3\_RX\_BCLK** 0x401F8050U, 0x1U, 0, 0, 0x401F8100U
- #define **IOMUXC\_GPIO\_SD\_13\_ARM\_CM7\_TXEV** 0x401F8050U, 0x2U, 0, 0, 0x401F8100U
- #define **IOMUXC\_GPIO\_SD\_13\_CCM\_PMIC\_RDY** 0x401F8050U, 0x3U, 0, 0, 0x401F8100U
- #define **IOMUXC\_GPIO\_SD\_13\_FLEXIO1\_IO19** 0x401F8050U, 0x4U, 0, 0, 0x401F8100U
- #define **IOMUXC\_GPIO\_SD\_13\_GPIO2\_IO13** 0x401F8050U, 0x5U, 0, 0, 0x401F8100U
- #define **IOMUXC\_GPIO\_SD\_13\_SRC\_BT\_CFG03** 0x401F8050U, 0x6U, 0, 0, 0x401F8100U

- #define **IOMUXC\_GPIO\_SD\_12\_FLEXSPI\_A\_DQS** 0x401F8054U, 0x0U, 0x401F8194U, 0x1U, 0x401F8104U
- #define **IOMUXC\_GPIO\_SD\_12\_LPSPI2\_PCS0** 0x401F8054U, 0x1U, 0x401F81E0U, 0x1U, 0x401F8104U
- #define **IOMUXC\_GPIO\_SD\_12\_LPUART1\_TXD** 0x401F8054U, 0x2U, 0x401F81F4U, 0x0U, 0x401F8104U
- #define **IOMUXC\_GPIO\_SD\_12\_FLEXIO1\_IO18** 0x401F8054U, 0x4U, 0, 0, 0x401F8104U
- #define **IOMUXC\_GPIO\_SD\_12\_GPIO2\_IO12** 0x401F8054U, 0x5U, 0, 0, 0x401F8104U
- #define **IOMUXC\_GPIO\_SD\_12\_WDOG2\_RST\_B\_DEB** 0x401F8054U, 0x6U, 0, 0, 0x401F8104U
- #define **IOMUXC\_GPIO\_SD\_11\_FLEXSPI\_A\_DATA3** 0x401F8058U, 0x0U, 0, 0, 0x401F8108U
- #define **IOMUXC\_GPIO\_SD\_11\_LPSPI2\_SCK** 0x401F8058U, 0x1U, 0x401F81E4U, 0x1U, 0x401F8108U
- #define **IOMUXC\_GPIO\_SD\_11\_LPUART1\_RXD** 0x401F8058U, 0x2U, 0x401F81F0U, 0x0U, 0x401F8108U
- #define **IOMUXC\_GPIO\_SD\_11\_FLEXIO1\_IO17** 0x401F8058U, 0x4U, 0, 0, 0x401F8108U
- #define **IOMUXC\_GPIO\_SD\_11\_GPIO2\_IO11** 0x401F8058U, 0x5U, 0, 0, 0x401F8108U
- #define **IOMUXC\_GPIO\_SD\_11\_WDOG1\_RST\_B\_DEB** 0x401F8058U, 0x6U, 0, 0, 0x401F8108U
- #define **IOMUXC\_GPIO\_SD\_10\_FLEXSPI\_A\_SCLK** 0x401F805CU, 0x0U, 0, 0, 0x401F810CU
- #define **IOMUXC\_GPIO\_SD\_10\_LPSPI2\_SDO** 0x401F805CU, 0x1U, 0x401F81ECU, 0x1U, 0x401F810CU
- #define **IOMUXC\_GPIO\_SD\_10\_LPUART2\_TXD** 0x401F805CU, 0x2U, 0x401F81FCU, 0x1U, 0x401F810CU
- #define **IOMUXC\_GPIO\_SD\_10\_FLEXIO1\_IO16** 0x401F805CU, 0x4U, 0, 0, 0x401F810CU
- #define **IOMUXC\_GPIO\_SD\_10\_GPIO2\_IO10** 0x401F805CU, 0x5U, 0, 0, 0x401F810CU
- #define **IOMUXC\_GPIO\_SD\_09\_FLEXSPI\_A\_DATA0** 0x401F8060U, 0x0U, 0, 0, 0x401F8110U
- #define **IOMUXC\_GPIO\_SD\_09\_LPSPI2\_SDI** 0x401F8060U, 0x1U, 0x401F81E8U, 0x1U, 0x401F8110U
- #define **IOMUXC\_GPIO\_SD\_09\_LPUART2\_RXD** 0x401F8060U, 0x2U, 0x401F81F8U, 0x0U, 0x401F8110U
- #define **IOMUXC\_GPIO\_SD\_09\_FLEXIO1\_IO15** 0x401F8060U, 0x4U, 0, 0, 0x401F8110U
- #define **IOMUXC\_GPIO\_SD\_09\_GPIO2\_IO09** 0x401F8060U, 0x5U, 0, 0, 0x401F8110U
- #define **IOMUXC\_GPIO\_SD\_08\_FLEXSPI\_A\_DATA2** 0x401F8064U, 0x0U, 0, 0, 0x401F8114U
- #define **IOMUXC\_GPIO\_SD\_08\_LPI2C2\_SCL** 0x401F8064U, 0x1U, 0x401F81C8U, 0x2U, 0x401F8114U
- #define **IOMUXC\_GPIO\_SD\_08\_LPSPI1\_SCK** 0x401F8064U, 0x2U, 0x401F81D4U, 0x1U, 0x401F8114U
- #define **IOMUXC\_GPIO\_SD\_08\_FLEXIO1\_IO14** 0x401F8064U, 0x4U, 0, 0, 0x401F8114U
- #define **IOMUXC\_GPIO\_SD\_08\_GPIO2\_IO08** 0x401F8064U, 0x5U, 0, 0, 0x401F8114U
- #define **IOMUXC\_GPIO\_SD\_07\_FLEXSPI\_A\_DATA1** 0x401F8068U, 0x0U, 0, 0, 0x401F8118U
- #define **IOMUXC\_GPIO\_SD\_07\_LPI2C2\_SDA** 0x401F8068U, 0x1U, 0x401F81CCU, 0x2U, 0x401F8118U
- #define **IOMUXC\_GPIO\_SD\_07\_LPSPI1\_PCS0** 0x401F8068U, 0x2U, 0x401F81D0U, 0x1U, 0x401F8118U
- #define **IOMUXC\_GPIO\_SD\_07\_FLEXIO1\_IO13** 0x401F8068U, 0x4U, 0, 0, 0x401F8118U

- #define **IOMUXC\_GPIO\_SD\_07\_GPIO2\_IO07** 0x401F8068U, 0x5U, 0, 0, 0x401F8118U
- #define **IOMUXC\_GPIO\_SD\_06\_FLEXSPI\_A\_SS0\_B** 0x401F806CU, 0x0U, 0, 0, 0x401F811-CU
- #define **IOMUXC\_GPIO\_SD\_06\_LPI2C1\_SCL** 0x401F806CU, 0x1U, 0x401F81C0U, 0x1U, 0x401F811CU
- #define **IOMUXC\_GPIO\_SD\_06\_LPSP1\_SDO** 0x401F806CU, 0x2U, 0x401F81DCU, 0x1U, 0x401F811CU
- #define **IOMUXC\_GPIO\_SD\_06\_FLEXIO1\_IO12** 0x401F806CU, 0x4U, 0, 0, 0x401F811CU
- #define **IOMUXC\_GPIO\_SD\_06\_GPIO2\_IO06** 0x401F806CU, 0x5U, 0, 0, 0x401F811CU
- #define **IOMUXC\_GPIO\_SD\_05\_FLEXSPI\_A\_SS1\_B** 0x401F8070U, 0x0U, 0, 0, 0x401F8120-U
- #define **IOMUXC\_GPIO\_SD\_05\_LPI2C1\_SDA** 0x401F8070U, 0x1U, 0x401F81C4U, 0x1U, 0x401F8120U
- #define **IOMUXC\_GPIO\_SD\_05\_LPSP1\_SDI** 0x401F8070U, 0x2U, 0x401F81D8U, 0x1U, 0x401F8120U
- #define **IOMUXC\_GPIO\_SD\_05\_FLEXIO1\_IO11** 0x401F8070U, 0x4U, 0, 0, 0x401F8120U
- #define **IOMUXC\_GPIO\_SD\_05\_GPIO2\_IO05** 0x401F8070U, 0x5U, 0, 0, 0x401F8120U
- #define **IOMUXC\_GPIO\_SD\_04\_FLEXPWM1\_PWM1\_A** 0x401F8074U, 0x2U, 0x401F8178-U, 0x0U, 0x401F8124U
- #define **IOMUXC\_GPIO\_SD\_04\_CCM\_WAIT** 0x401F8074U, 0x3U, 0, 0, 0x401F8124U
- #define **IOMUXC\_GPIO\_SD\_04\_FLEXIO1\_IO10** 0x401F8074U, 0x4U, 0, 0, 0x401F8124U
- #define **IOMUXC\_GPIO\_SD\_04\_GPIO2\_IO04** 0x401F8074U, 0x5U, 0, 0, 0x401F8124U
- #define **IOMUXC\_GPIO\_SD\_04\_SRC\_BOOT\_MODE00** 0x401F8074U, 0x6U, 0, 0, 0x401F8124U
- #define **IOMUXC\_GPIO\_SD\_04\_FLEXSPI\_B\_DATA03** 0x401F8074U, 0x0U, 0, 0, 0x401F8124U
- #define **IOMUXC\_GPIO\_SD\_04\_SAI3\_RX\_SYNC** 0x401F8074U, 0x1U, 0, 0, 0x401F8124U
- #define **IOMUXC\_GPIO\_SD\_03\_FLEXSPI\_B\_DATA00** 0x401F8078U, 0x0U, 0, 0, 0x401F8128U
- #define **IOMUXC\_GPIO\_SD\_03\_SAI3\_RX\_DATA** 0x401F8078U, 0x1U, 0, 0, 0x401F8128U
- #define **IOMUXC\_GPIO\_SD\_03\_FLEXPWM1\_PWM1\_B** 0x401F8078U, 0x2U, 0x401F8188-U, 0x0U, 0x401F8128U
- #define **IOMUXC\_GPIO\_SD\_03\_CCM\_REF\_EN\_B** 0x401F8078U, 0x3U, 0, 0, 0x401F8128U
- #define **IOMUXC\_GPIO\_SD\_03\_FLEXIO1\_IO09** 0x401F8078U, 0x4U, 0, 0, 0x401F8128U
- #define **IOMUXC\_GPIO\_SD\_03\_GPIO2\_IO03** 0x401F8078U, 0x5U, 0, 0, 0x401F8128U
- #define **IOMUXC\_GPIO\_SD\_03\_SRC\_BOOT\_MODE01** 0x401F8078U, 0x6U, 0, 0, 0x401F8128U
- #define **IOMUXC\_GPIO\_SD\_02\_FLEXSPI\_B\_DATA02** 0x401F807CU, 0x0U, 0, 0, 0x401F812CU
- #define **IOMUXC\_GPIO\_SD\_02\_SAI3\_TX\_DATA** 0x401F807CU, 0x1U, 0, 0, 0x401F812CU
- #define **IOMUXC\_GPIO\_SD\_02\_FLEXPWM1\_PWM0\_A** 0x401F807CU, 0x2U, 0x401F8174-U, 0x0U, 0x401F812CU
- #define **IOMUXC\_GPIO\_SD\_02\_CCM\_CLKO1** 0x401F807CU, 0x3U, 0, 0, 0x401F812CU
- #define **IOMUXC\_GPIO\_SD\_02\_FLEXIO1\_IO08** 0x401F807CU, 0x4U, 0, 0, 0x401F812CU
- #define **IOMUXC\_GPIO\_SD\_02\_GPIO2\_IO02** 0x401F807CU, 0x5U, 0, 0, 0x401F812CU
- #define **IOMUXC\_GPIO\_SD\_02\_SRC\_BT\_CFG00** 0x401F807CU, 0x6U, 0, 0, 0x401F812CU
- #define **IOMUXC\_GPIO\_SD\_01\_FLEXSPI\_B\_DATA01** 0x401F8080U, 0x0U, 0, 0, 0x401F8130U
- #define **IOMUXC\_GPIO\_SD\_01\_SAI3\_TX\_BCLK** 0x401F8080U, 0x1U, 0, 0, 0x401F8130U
- #define **IOMUXC\_GPIO\_SD\_01\_FLEXPWM1\_PWM0\_B** 0x401F8080U, 0x2U, 0x401F8184-U, 0x0U, 0x401F8130U
- #define **IOMUXC\_GPIO\_SD\_01\_CCM\_CLKO2** 0x401F8080U, 0x3U, 0, 0, 0x401F8130U

- #define **IOMUXC\_GPIO\_SD\_01\_FLEXIO1\_IO07** 0x401F8080U, 0x4U, 0, 0, 0x401F8130U
- #define **IOMUXC\_GPIO\_SD\_01\_GPIO2\_IO01** 0x401F8080U, 0x5U, 0, 0, 0x401F8130U
- #define **IOMUXC\_GPIO\_SD\_01\_SRC\_BT\_CFG01** 0x401F8080U, 0x6U, 0, 0, 0x401F8130U
- #define **IOMUXC\_GPIO\_SD\_00\_FLEXSPI\_B\_SS0\_B** 0x401F8084U, 0x0U, 0, 0, 0x401F8134U
- #define **IOMUXC\_GPIO\_SD\_00\_SAI3\_TX\_SYNC** 0x401F8084U, 0x1U, 0, 0, 0x401F8134U
- #define **IOMUXC\_GPIO\_SD\_00\_ARM\_CM7\_RXEV** 0x401F8084U, 0x2U, 0x401F8220U, 0x1U, 0x401F8134U
- #define **IOMUXC\_GPIO\_SD\_00\_CCM\_STOP** 0x401F8084U, 0x3U, 0, 0, 0x401F8134U
- #define **IOMUXC\_GPIO\_SD\_00\_FLEXIO1\_IO06** 0x401F8084U, 0x4U, 0, 0, 0x401F8134U
- #define **IOMUXC\_GPIO\_SD\_00\_GPIO2\_IO00** 0x401F8084U, 0x5U, 0, 0, 0x401F8134U
- #define **IOMUXC\_GPIO\_SD\_00\_SRC\_BT\_CFG02** 0x401F8084U, 0x6U, 0, 0, 0x401F8134U
- #define **IOMUXC\_GPIO\_13\_LPUART2\_RXD** 0x401F8088U, 0x0U, 0x401F81F8U, 0x1U, 0x401F8138U
- #define **IOMUXC\_GPIO\_13\_LPSPi2\_PCS2** 0x401F8088U, 0x1U, 0, 0, 0x401F8138U
- #define **IOMUXC\_GPIO\_13\_KPP\_ROW03** 0x401F8088U, 0x2U, 0x401F81B8U, 0x0U, 0x401F8138U
- #define **IOMUXC\_GPIO\_13\_USB\_OTG1\_ID** 0x401F8088U, 0x3U, 0x401F8170U, 0x1U, 0x401F8138U
- #define **IOMUXC\_GPIO\_13\_FLEXIO1\_IO05** 0x401F8088U, 0x4U, 0, 0, 0x401F8138U
- #define **IOMUXC\_GPIO\_13\_GPIOMUX\_IO13** 0x401F8088U, 0x5U, 0, 0, 0x401F8138U
- #define **IOMUXC\_GPIO\_13\_SPDIF\_LOCK** 0x401F8088U, 0x6U, 0, 0, 0x401F8138U
- #define **IOMUXC\_GPIO\_13\_ARM\_TRACE1** 0x401F8088U, 0x7U, 0, 0, 0x401F8138U
- #define **IOMUXC\_GPIO\_12\_LPUART3\_TXD** 0x401F808CU, 0x0U, 0x401F8204U, 0x1U, 0x401F813CU
- #define **IOMUXC\_GPIO\_12\_LPI2C1\_SCL** 0x401F808CU, 0x1U, 0x401F81C0U, 0x2U, 0x401F813CU
- #define **IOMUXC\_GPIO\_12\_KPP\_COL00** 0x401F808CU, 0x2U, 0x401F819CU, 0x1U, 0x401F813CU
- #define **IOMUXC\_GPIO\_12\_USB\_OTG1\_OC** 0x401F808CU, 0x3U, 0x401F821CU, 0x1U, 0x401F813CU
- #define **IOMUXC\_GPIO\_12\_FLEXIO1\_IO04** 0x401F808CU, 0x4U, 0, 0, 0x401F813CU
- #define **IOMUXC\_GPIO\_12\_GPIOMUX\_IO12** 0x401F808CU, 0x5U, 0, 0, 0x401F813CU
- #define **IOMUXC\_GPIO\_12\_SPDIF\_EXT\_CLK** 0x401F808CU, 0x6U, 0x401F8218U, 0x0U, 0x401F813CU
- #define **IOMUXC\_GPIO\_12\_ARM\_TRACE2** 0x401F808CU, 0x7U, 0, 0, 0x401F813CU
- #define **IOMUXC\_GPIO\_11\_LPUART3\_RXD** 0x401F8090U, 0x0U, 0x401F8200U, 0x1U, 0x401F8140U
- #define **IOMUXC\_GPIO\_11\_LPI2C1\_SDA** 0x401F8090U, 0x1U, 0x401F81C4U, 0x2U, 0x401F8140U
- #define **IOMUXC\_GPIO\_11\_KPP\_ROW00** 0x401F8090U, 0x2U, 0x401F81ACU, 0x1U, 0x401F8140U
- #define **IOMUXC\_GPIO\_11\_FLEXSPI\_B\_SS1\_B** 0x401F8090U, 0x3U, 0, 0, 0x401F8140U
- #define **IOMUXC\_GPIO\_11\_FLEXIO1\_IO03** 0x401F8090U, 0x4U, 0, 0, 0x401F8140U
- #define **IOMUXC\_GPIO\_11\_GPIOMUX\_IO11** 0x401F8090U, 0x5U, 0, 0, 0x401F8140U
- #define **IOMUXC\_GPIO\_11\_SPDIF\_OUT** 0x401F8090U, 0x6U, 0, 0, 0x401F8140U
- #define **IOMUXC\_GPIO\_11\_ARM\_TRACE3** 0x401F8090U, 0x7U, 0, 0, 0x401F8140U
- #define **IOMUXC\_GPIO\_10\_LPUART1\_TXD** 0x401F8094U, 0x0U, 0x401F81F4U, 0x1U, 0x401F8144U
- #define **IOMUXC\_GPIO\_10\_LPI2C1\_HREQ** 0x401F8094U, 0x1U, 0x401F81BCU, 0x1U, 0x401F8144U
- #define **IOMUXC\_GPIO\_10\_EWM\_OUT\_B** 0x401F8094U, 0x2U, 0, 0, 0x401F8144U
- #define **IOMUXC\_GPIO\_10\_LPI2C2\_SCL** 0x401F8094U, 0x3U, 0x401F81C8U, 0x3U, 0x401F8144U

F8144U

- #define **IOMUXC\_GPIO\_10\_FLEXIO1\_IO02** 0x401F8094U, 0x4U, 0, 0, 0x401F8144U
- #define **IOMUXC\_GPIO\_10\_GPIOMUX\_IO10** 0x401F8094U, 0x5U, 0, 0, 0x401F8144U
- #define **IOMUXC\_GPIO\_10\_SPDIF\_IN** 0x401F8094U, 0x6U, 0x401F8214U, 0x0U, 0x401F8144U
- #define **IOMUXC\_GPIO\_09\_LPUART1\_RXD** 0x401F8098U, 0x0U, 0x401F81F0U, 0x1U, 0x401F8148U
- #define **IOMUXC\_GPIO\_09\_WDOG1\_B** 0x401F8098U, 0x1U, 0, 0, 0x401F8148U
- #define **IOMUXC\_GPIO\_09\_FLEXSPI\_A\_SS1\_B** 0x401F8098U, 0x2U, 0, 0, 0x401F8148U
- #define **IOMUXC\_GPIO\_09\_LPI2C2\_SDA** 0x401F8098U, 0x3U, 0x401F81CCU, 0x3U, 0x401F8148U
- #define **IOMUXC\_GPIO\_09\_FLEXIO1\_IO01** 0x401F8098U, 0x4U, 0, 0, 0x401F8148U
- #define **IOMUXC\_GPIO\_09\_GPIOMUX\_IO09** 0x401F8098U, 0x5U, 0, 0, 0x401F8148U
- #define **IOMUXC\_GPIO\_09\_SPDIF\_SR\_CLK** 0x401F8098U, 0x6U, 0, 0, 0x401F8148U
- #define **IOMUXC\_GPIO\_08\_SAI1\_MCLK** 0x401F809CU, 0x0U, 0, 0, 0x401F814CU
- #define **IOMUXC\_GPIO\_08\_GPT1\_CLK** 0x401F809CU, 0x1U, 0, 0, 0x401F814CU
- #define **IOMUXC\_GPIO\_08\_FLEXPWM1\_PWM3\_A** 0x401F809CU, 0x2U, 0x401F8180U, 0x1U, 0x401F814CU
- #define **IOMUXC\_GPIO\_08\_LPUART3\_TXD** 0x401F809CU, 0x3U, 0x401F8204U, 0x2U, 0x401F814CU
- #define **IOMUXC\_GPIO\_08\_FLEXIO1\_IO00** 0x401F809CU, 0x4U, 0, 0, 0x401F814CU
- #define **IOMUXC\_GPIO\_08\_GPIOMUX\_IO08** 0x401F809CU, 0x5U, 0, 0, 0x401F814CU
- #define **IOMUXC\_GPIO\_08\_LPUART1\_CTS\_B** 0x401F809CU, 0x6U, 0, 0, 0x401F814CU
- #define **IOMUXC\_GPIO\_07\_SAI1\_TX\_SYNC** 0x401F80A0U, 0x0U, 0, 0, 0x401F8150U
- #define **IOMUXC\_GPIO\_07\_GPT1\_COMPARE1** 0x401F80A0U, 0x1U, 0, 0, 0x401F8150U
- #define **IOMUXC\_GPIO\_07\_FLEXPWM1\_PWM3\_B** 0x401F80A0U, 0x2U, 0x401F8190U, 0x1U, 0x401F8150U
- #define **IOMUXC\_GPIO\_07\_LPUART3\_RXD** 0x401F80A0U, 0x3U, 0x401F8200U, 0x2U, 0x401F8150U
- #define **IOMUXC\_GPIO\_07\_SPDIF\_LOCK** 0x401F80A0U, 0x4U, 0, 0, 0x401F8150U
- #define **IOMUXC\_GPIO\_07\_GPIOMUX\_IO07** 0x401F80A0U, 0x5U, 0, 0, 0x401F8150U
- #define **IOMUXC\_GPIO\_07\_LPUART1\_RTS\_B** 0x401F80A0U, 0x6U, 0, 0, 0x401F8150U
- #define **IOMUXC\_GPIO\_06\_SAI1\_TX\_BCLK** 0x401F80A4U, 0x0U, 0, 0, 0x401F8154U
- #define **IOMUXC\_GPIO\_06\_GPT1\_CAPTURE1** 0x401F80A4U, 0x1U, 0, 0, 0x401F8154U
- #define **IOMUXC\_GPIO\_06\_FLEXPWM1\_PWM2\_A** 0x401F80A4U, 0x2U, 0x401F817CU, 0x1U, 0x401F8154U
- #define **IOMUXC\_GPIO\_06\_LPUART4\_TXD** 0x401F80A4U, 0x3U, 0x401F820CU, 0x1U, 0x401F8154U
- #define **IOMUXC\_GPIO\_06\_SPDIF\_EXT\_CLK** 0x401F80A4U, 0x4U, 0x401F8218U, 0x1U, 0x401F8154U
- #define **IOMUXC\_GPIO\_06\_GPIOMUX\_IO06** 0x401F80A4U, 0x5U, 0, 0, 0x401F8154U
- #define **IOMUXC\_GPIO\_05\_SAI1\_TX\_DATA01** 0x401F80A8U, 0x0U, 0, 0, 0x401F8158U
- #define **IOMUXC\_GPIO\_05\_GPT1\_COMPARE2** 0x401F80A8U, 0x1U, 0, 0, 0x401F8158U
- #define **IOMUXC\_GPIO\_05\_FLEXPWM1\_PWM2\_B** 0x401F80A8U, 0x2U, 0x401F818CU, 0x1U, 0x401F8158U
- #define **IOMUXC\_GPIO\_05\_LPUART4\_RXD** 0x401F80A8U, 0x3U, 0x401F8208U, 0x1U, 0x401F8158U
- #define **IOMUXC\_GPIO\_05\_SPDIF\_OUT** 0x401F80A8U, 0x4U, 0, 0, 0x401F8158U
- #define **IOMUXC\_GPIO\_05\_GPIOMUX\_IO05** 0x401F80A8U, 0x5U, 0, 0, 0x401F8158U
- #define **IOMUXC\_GPIO\_04\_SAI1\_TX\_DATA00** 0x401F80ACU, 0x0U, 0, 0, 0x401F815CU
- #define **IOMUXC\_GPIO\_04\_GPT1\_CAPTURE2** 0x401F80ACU, 0x1U, 0, 0, 0x401F815CU
- #define **IOMUXC\_GPIO\_04\_FLEXPWM1\_PWM1\_A** 0x401F80ACU, 0x2U, 0x401F8178U, 0x1U, 0x401F815CU

- #define **IOMUXC\_GPIO\_04\_SPDIF\_IN** 0x401F80ACU, 0x4U, 0x401F8214U, 0x1U, 0x401F815CU
- #define **IOMUXC\_GPIO\_04\_GPIOMUX\_IO04** 0x401F80ACU, 0x5U, 0, 0, 0x401F815CU
- #define **IOMUXC\_GPIO\_03\_SAI1\_RX\_DATA00** 0x401F80B0U, 0x0U, 0, 0, 0x401F8160U
- #define **IOMUXC\_GPIO\_03\_GPT1\_COMPARE3** 0x401F80B0U, 0x1U, 0, 0, 0x401F8160U
- #define **IOMUXC\_GPIO\_03\_FLEXPWM1\_PWM1\_B** 0x401F80B0U, 0x2U, 0x401F8188U, 0x1U, 0x401F8160U
- #define **IOMUXC\_GPIO\_03\_SPDIF\_SR\_CLK** 0x401F80B0U, 0x4U, 0, 0, 0x401F8160U
- #define **IOMUXC\_GPIO\_03\_GPIOMUX\_IO03** 0x401F80B0U, 0x5U, 0, 0, 0x401F8160U
- #define **IOMUXC\_GPIO\_02\_SAI1\_RX\_SYNC** 0x401F80B4U, 0x0U, 0, 0, 0x401F8164U
- #define **IOMUXC\_GPIO\_02\_WDOG2\_B** 0x401F80B4U, 0x1U, 0, 0, 0x401F8164U
- #define **IOMUXC\_GPIO\_02\_FLEXPWM1\_PWM0\_A** 0x401F80B4U, 0x2U, 0x401F8174U, 0x1U, 0x401F8164U
- #define **IOMUXC\_GPIO\_02\_LPI2C1\_SCL** 0x401F80B4U, 0x3U, 0x401F81C0U, 0x3U, 0x401F8164U
- #define **IOMUXC\_GPIO\_02\_KPP\_COL03** 0x401F80B4U, 0x4U, 0x401F81A8U, 0x1U, 0x401F8164U
- #define **IOMUXC\_GPIO\_02\_GPIOMUX\_IO02** 0x401F80B4U, 0x5U, 0, 0, 0x401F8164U
- #define **IOMUXC\_GPIO\_01\_SAI1\_RX\_BCLK** 0x401F80B8U, 0x0U, 0, 0, 0x401F8168U
- #define **IOMUXC\_GPIO\_01\_WDOG1\_ANY** 0x401F80B8U, 0x1U, 0, 0, 0x401F8168U
- #define **IOMUXC\_GPIO\_01\_FLEXPWM1\_PWM0\_B** 0x401F80B8U, 0x2U, 0x401F8184U, 0x1U, 0x401F8168U
- #define **IOMUXC\_GPIO\_01\_LPI2C1\_SDA** 0x401F80B8U, 0x3U, 0x401F81C4U, 0x3U, 0x401F8168U
- #define **IOMUXC\_GPIO\_01\_KPP\_ROW03** 0x401F80B8U, 0x4U, 0x401F81B8U, 0x1U, 0x401F8168U
- #define **IOMUXC\_GPIO\_01\_GPIOMUX\_IO01** 0x401F80B8U, 0x5U, 0, 0, 0x401F8168U
- #define **IOMUXC\_GPIO\_00\_FLEXSPI\_B\_DQS** 0x401F80BCU, 0x0U, 0x401F8198U, 0x1U, 0x401F816CU
- #define **IOMUXC\_GPIO\_00\_SAI3\_MCLK** 0x401F80BCU, 0x1U, 0, 0, 0x401F816CU
- #define **IOMUXC\_GPIO\_00\_LPSPi2\_PCS3** 0x401F80BCU, 0x2U, 0, 0, 0x401F816CU
- #define **IOMUXC\_GPIO\_00\_LPSPi1\_PCS3** 0x401F80BCU, 0x3U, 0, 0, 0x401F816CU
- #define **IOMUXC\_GPIO\_00\_PIT\_TRIGGER00** 0x401F80BCU, 0x4U, 0, 0, 0x401F816CU
- #define **IOMUXC\_GPIO\_00\_GPIOMUX\_IO00** 0x401F80BCU, 0x5U, 0, 0, 0x401F816CU
- #define **IOMUXC\_SNVS\_PMIC\_ON\_REQ\_SNVS\_LP\_PMIC\_ON\_REQ** 0x400A8000U, 0x0U, 0, 0, 0x400A8010U
- #define **IOMUXC\_SNVS\_PMIC\_ON\_REQ\_GPIO5\_IO00** 0x400A8000U, 0x5U, 0, 0, 0x400A8010U
- #define **IOMUXC\_SNVS\_TEST\_MODE** 0, 0, 0, 0, 0x400A8004U
- #define **IOMUXC\_SNVS POR\_B** 0, 0, 0, 0, 0x400A8008U
- #define **IOMUXC\_SNVS\_ONOFF** 0, 0, 0, 0, 0x400A800CU

## 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\_MQSEEnterSoftwareReset** (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.*

## 5.2 Macro Definition Documentation

### 5.2.1 #define FSL\_IOMUXC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 4))

## 5.3 Function Documentation

### 5.3.1 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.   |

### 5.3.2 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\_SCLS:

```
* IOMUXC_SetPinConfig(IOMUXC_PTA3_LPI2C0_SCLS, 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.      |

### 5.3.3 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.                                            |

**5.3.4 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. |

**5.3.5 static void IOMUXC\_MQSEEnterSoftwareReset ( 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. |

**5.3.6 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.   |

**5.3.7 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 6

## ADC\_ETC: ADC External Trigger Control

### 6.1 Overview

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

### 6.2 Typical use case

#### 6.2.1 Software trigger Configuration

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

#### 6.2.2 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.*

- enum `adc_etc_dma_mode_selection_t`  
*DMA mode selection.*

## 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 falgs.*
- 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 falgs.*
- static void `ADC_ETC_ClearDMAStatusFlags` (ADC\_ETC\_Type \*base, uint32\_t mask)  
*Clear the DMA request status falgs.*
- 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_GetADCConversionValue` (ADC\_ETC\_Type \*base, uint32\_t triggerGroup, uint32\_t chainGroup)  
*Get ADC conversion result from external XBAR sources.*

## 6.3 Data Structure Documentation

### 6.3.1 struct `adc_etc_config_t`

### 6.3.2 struct `adc_etc_trigger_chain_config_t`

### 6.3.3 struct `adc_etc_trigger_config_t`

## 6.4 Macro Definition Documentation

**6.4.1 #define FSL\_ADC\_ETC\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 1))**

Version 2.2.1.

**6.4.2 #define ADC\_ETC\_DMA\_CTRL\_TRGn\_REQ\_MASK 0xFF0000U****6.5 Function Documentation****6.5.1 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. |

**6.5.2 void ADC\_ETC\_Deinit ( ADC\_ETC\_Type \* *base* )**

Parameters

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

**6.5.3 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. |
|---------------|------------------------------------------|

#### 6.5.4 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. |

#### 6.5.5 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 configurated.

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. |

#### 6.5.6 **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".

### 6.5.7 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". |

### 6.5.8 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. |

### 6.5.9 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. |

### 6.5.10 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.

### 6.5.11 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. |

### 6.5.12 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. |

### 6.5.13 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. |

#### 6.5.14 `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 7

## AIPSTZ: AHB to IP Bridge

### 7.1 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_SetMasterPriviledgeLevel` (`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.*

### 7.2 Enumeration Type Documentation

#### 7.2.1 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.

## 7.2.2 enum aipstz\_master\_t

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

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

## 7.2.4 enum aipstz\_peripheral\_t

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

## 7.3 Function Documentation

### 7.3.1 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> . |

### 7.3.2 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 8

## AOI: Crossbar AND/OR/INVERT Driver

### 8.1 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

### 8.2 Function groups

#### 8.2.1 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.

#### 8.2.2 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.

### 8.3 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.*

## 8.4 Data Structure Documentation

### 8.4.1 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.*

## 8.5 Macro Definition Documentation

### 8.5.1 `#define FSL_AOI_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))`

## 8.6 Enumeration Type Documentation

### 8.6.1 `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.

### 8.6.2 `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.

## 8.7 Function Documentation

### 8.7.1 `void AOI_Init ( AOI_Type * base )`

This function un-gates the AOI clock.

Parameters

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

### 8.7.2 `void AOI_Deinit ( AOI_Type * base )`

This function shutdowns AOI module.

## Parameters

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

### 8.7.3 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 .                            |

### 8.7.4 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 aoi_event_t.                                                                                                           |
| <i>eventConfig</i> | Pointer to type aoi_event_config_t structure. The user is responsible for filling out the members of this structure and passing the pointer to this function. |

# **Chapter 9**

## **CACHE: ARMV7-M7 CACHE Memory Controller**

### **9.1 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.

### **9.2 Function groups**

#### **9.2.1 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.

#### **9.2.2 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 its 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 pI310)

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.*

## 9.3 Macro Definition Documentation

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

## 9.4 Function Documentation

9.4.1 **void L1CACHE\_InvalidateCacheByRange ( 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\_BYT-E) 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.

#### 9.4.2 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.

#### 9.4.3 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.

9.4.4 **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.

#### 9.4.5 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.

#### 9.4.6 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.

**9.4.7 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.

**9.4.8 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 10

## Common Driver

### 10.1 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_USACARD` = 58,
   
`kStatusGroup_LPC_DMIC` = 58,
   
`kStatusGroup_SDIF` = 59,

```

kStatusGroup_POWER_MANAGER = 159 }

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

Generic status return codes.

```

## Functions

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

## Driver version

- #define **FSL\_COMMON\_DRIVER\_VERSION** (MAKE\_VERSION(2, 3, 1))  
*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()**

## 10.2 Macro Definition Documentation

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

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

### 10.2.3 #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 |

### 10.2.4 #define FSL\_COMMON\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 1))

### 10.2.5 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_NONE 0U

### 10.2.6 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_UART 1U

### 10.2.7 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_LPUART 2U

### 10.2.8 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_LPSCI 3U

### 10.2.9 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_USBCDC 4U

### 10.2.10 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_FLEXCOMM 5U

### 10.2.11 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_IUART 6U

### 10.2.12 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_VUSART 7U

### 10.2.13 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_MINI\_USART 8U

### 10.2.14 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_SWO 9U

### 10.2.15 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_QSCI 10U

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

## 10.3 Typedef Documentation

### 10.3.1 typedef int32\_t status\_t

## 10.4 Enumeration Type Documentation

### 10.4.1 enum \_status\_groups

Enumerator

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

*kStatusGroup\_FLEXCAN* Group number for FlexCAN status codes.  
*kStatusGroup\_LTC* Group number for LTC status codes.  
*kStatusGroup\_FLEXIO\_CAMERA* Group number for FLEXIO CAMERA status codes.  
*kStatusGroup\_LPC\_SPI* Group number for LPC\_SPI status codes.  
*kStatusGroup\_LPC\_USART* Group number for LPC\_USART status codes.  
*kStatusGroup\_DMIC* Group number for DMIC status codes.  
*kStatusGroup\_SDIF* Group number for SDIF status codes.  
*kStatusGroup\_SPIFI* Group number for SPIFI status codes.  
*kStatusGroup OTP* Group number for OTP status codes.  
*kStatusGroup\_MCAN* Group number for MCAN status codes.  
*kStatusGroup\_CAAM* Group number for CAAM status codes.  
*kStatusGroup\_ECSPI* Group number for ECSPI status codes.  
*kStatusGroup\_USDHC* Group number for USDHC status codes.  
*kStatusGroup\_LPC\_I2C* Group number for LPC\_I2C status codes.  
*kStatusGroup\_DCP* Group number for DCP status codes.  
*kStatusGroup\_MSCAN* Group number for MSCAN status codes.  
*kStatusGroup\_ESAI* Group number for ESAI status codes.  
*kStatusGroup\_FLEXSPI* Group number for FLEXSPI status codes.  
*kStatusGroup\_MMDC* Group number for MMDC status codes.  
*kStatusGroup\_PDM* Group number for MIC status codes.  
*kStatusGroup\_SDMA* Group number for SDMA status codes.  
*kStatusGroup\_ICS* Group number for ICS status codes.  
*kStatusGroup\_SPDIF* Group number for SPDIF status codes.  
*kStatusGroup\_LPC\_MINISPI* Group number for LPC\_MINISPI status codes.  
*kStatusGroup\_HASHCRYPT* Group number for Hashcrypt status codes.  
*kStatusGroup\_LPC\_SPI\_SSP* Group number for LPC\_SPI\_SSP status codes.  
*kStatusGroup\_I3C* Group number for I3C status codes.  
*kStatusGroup\_LPC\_I2C\_1* Group number for LPC\_I2C\_1 status codes.  
*kStatusGroup\_NOTIFIER* Group number for NOTIFIER status codes.  
*kStatusGroup\_DebugConsole* Group number for debug console status codes.  
*kStatusGroup\_SEMC* Group number for SEMC status codes.  
*kStatusGroup\_ApplicationRangeStart* Starting number for application groups.  
*kStatusGroup\_IAP* Group number for IAP status codes.  
*kStatusGroup\_SFA* Group number for SFA status codes.  
*kStatusGroup\_SPC* Group number for SPC status codes.  
*kStatusGroup\_PUF* Group number for PUF status codes.  
*kStatusGroup\_TOUCH\_PANEL* Group number for touch panel status codes.  
*kStatusGroup\_HAL\_GPIO* Group number for HAL GPIO status codes.  
*kStatusGroup\_HAL\_UART* Group number for HAL UART status codes.  
*kStatusGroup\_HAL\_TIMER* Group number for HAL TIMER status codes.  
*kStatusGroup\_HAL\_SPI* Group number for HAL SPI status codes.  
*kStatusGroup\_HAL\_I2C* Group number for HAL I2C status codes.  
*kStatusGroup\_HAL\_FLASH* Group number for HAL FLASH status codes.  
*kStatusGroup\_HAL\_PWM* Group number for HAL PWM status codes.  
*kStatusGroup\_HAL\_RNG* Group number for HAL RNG status codes.

*kStatusGroup\_HAL\_I2S* Group number for HAL I2S status codes.  
*kStatusGroup\_TIMERMANAGER* Group number for TiMER MANAGER status codes.  
*kStatusGroup\_SERIALMANAGER* Group number for SERIAL MANAGER status codes.  
*kStatusGroup\_LED* Group number for LED status codes.  
*kStatusGroup\_BUTTON* Group number for BUTTON status codes.  
*kStatusGroup\_EXTERN\_EEPROM* Group number for EXTERN EEPROM status codes.  
*kStatusGroup\_SHELL* Group number for SHELL status codes.  
*kStatusGroup\_MEM\_MANAGER* Group number for MEM MANAGER status codes.  
*kStatusGroup\_LIST* Group number for List status codes.  
*kStatusGroup\_OSA* Group number for OSA status codes.  
*kStatusGroup\_COMMON\_TASK* Group number for Common task status codes.  
*kStatusGroup\_MSG* Group number for messaging status codes.  
*kStatusGroup\_SDK\_OCOTP* Group number for OCOTP status codes.  
*kStatusGroup\_SDK\_FLEXSPINOR* Group number for FLEXSPINOR status codes.  
*kStatusGroup\_CODEC* Group number for codec status codes.  
*kStatusGroup\_ASRC* Group number for codec status ASRC.  
*kStatusGroup\_OTFAD* Group number for codec status codes.  
*kStatusGroup\_SDIOSLV* Group number for SDIOSLV status codes.  
*kStatusGroup\_MECC* Group number for MECC status codes.  
*kStatusGroup\_ENET\_QOS* Group number for ENET\_QOS status codes.  
*kStatusGroup\_LOG* Group number for LOG status codes.  
*kStatusGroup\_I3CBUS* Group number for I3CBUS status codes.  
*kStatusGroup\_QSCI* Group number for QSCI status codes.  
*kStatusGroup\_SNT* Group number for SNT status codes.  
*kStatusGroup\_QUEUEDSPI* Group number for QSPI status codes.  
*kStatusGroup\_POWER\_MANAGER* Group number for POWER\_MANAGER status codes.

#### 10.4.2 anonymous enum

Enumerator

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

#### 10.5 Function Documentation

### 10.5.1 **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. |
|------------|-------------------|

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

Parameters

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

### 10.5.3 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 11

## DCDC: DCDC Converter

### 11.1 Overview

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

### 11.2 Function groups

#### 11.2.1 Initialization and deinitialization

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

#### 11.2.2 Status

Provides functions to get the DCDC status.

#### 11.2.3 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.

## 11.3 Application guideline

### 11.3.1 Continous conduction mode

### 11.3.2 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)

- `void DCDC_SetLoopControlConfig (DCDC_Type *base, const dcdc_loop_control_config_t *config)`  
*Get the default setting for loop control configuration.*
- `void DCDC_SetMinPowerConfig (DCDC_Type *base, const dcdc_min_power_config_t *config)`  
*Configure the DCDC loop control.*
- `Configure for the min power.`
- `static void DCDC_SetLPCComparatorBiasValue (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(discontinous conduction mode).*
- `void DCDC_BootIntoCCM (DCDC_Type *base)`  
*Boot DCDC into CCM(continous conduction mode).*

## 11.4 Data Structure Documentation

### 11.4.1 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) `dcdc_over_current_threshold_t dcdc_detection_config_t::OverCurrentThreshold`
- (8) `dcdc_peak_current_threshold_t dcdc_detection_config_t::PeakCurrentThreshold`

#### 11.4.2 struct dc当地环控制配置

#### 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 dc当地环控制配置_t::enableCommonHysteresis`

This feature will improve transient supply ripple and efficiency.

- (2) `bool dc当地环控制配置_t::enableCommonThresholdDetection`
- (3) `bool dc当地环控制配置_t::enableInvertHysteresisSign`
- (4) `bool dc当地环控制配置_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.

**11.4.3 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**

**11.4.4 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.

**11.4.5 struct dc当地\_min\_power\_config\_t****Data Fields**

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

**Field Documentation**

- (1) `bool dc当地_min_power_config_t::enableUseHalfFreqForContinuous`

**11.5 Macro Definition Documentation****11.5.1 #define FSL\_DCDC\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 0))**

Version 2.3.0.

**11.6 Enumeration Type Documentation****11.6.1 enum \_dc当地\_status\_flags\_t**

Enumerator

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

**11.6.2 enum dc当地\_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.

### 11.6.3 enum dc当地\_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.

### 11.6.4 enum dc当地\_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.

### 11.6.5 enum dc当地\_count\_charging\_time\_period\_t

Enumerator

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

### 11.6.6 enum dc当地\_count\_charging\_time\_threshold\_t

Enumerator

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

### 11.6.7 enum dc当地\_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.

## 11.7 Function Documentation

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

Parameters

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

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

Parameters

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

### 11.7.3 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".

### 11.7.4 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.    |

### 11.7.5 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". |

**11.7.6 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" |
|---------------|----------------------------------------------------------------------|

**11.7.7 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" |

**11.7.8 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->enableAdjustHystericValue = false;
* config->countChargingTimePeriod = kDCDC_CountChargingTimePeriod8Cycle
 ;
* config->countChargingTimeThreshold = kDCDC_CountChargingTimeThreshold32
 ;
*
```

Parameters

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

### 11.7.9 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". |

### 11.7.10 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. |

### 11.7.11 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. |

### 11.7.12 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" |
|---------------|-------------------------------------------------------------------------|

**11.7.13 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". |

**11.7.14 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". |

**11.7.15 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". |

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

Parameters

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

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

**Deprecated** Do not use this function. It has been superceded 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. |

### 11.7.18 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. |

### 11.7.19 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. |

### 11.7.20 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". |

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

It is valid while zero cross detection is enabled. If ouput 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.    |

### 11.7.22 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. |
|-------------|-------------------------------|

### **11.7.23 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 12

## DCP: Data Co-Processor

### 12.1 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.

### 12.2 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.

## 12.3 Comments about API usage in RTOS

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

## 12.4 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 its operation with data (such as message, plaintext, ciphertext, or keys) all data should remain valid until the DCP operation completes.

## 12.5 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.

## 12.6 DCP Driver Examples

### 12.6.1 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, 6))`  
*DCP driver version.*

## 12.7 Data Structure Documentation

### 12.7.1 struct dcp\_work\_packet\_t

### 12.7.2 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`

### 12.7.3 struct dcp\_context\_t

### 12.7.4 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`

**12.8 Macro Definition Documentation****12.8.1 #define FSL\_DCP\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 6))**

Version 2.1.6.

Current version: 2.1.6

Change log:

- 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

**12.9 Enumeration Type Documentation**

### 12.9.1 enum \_dcp\_status

Enumerator

*kStatus\_DCP\_Again* Non-blocking function shall be called again.

### 12.9.2 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.

### 12.9.3 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.

### 12.9.4 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.

### 12.9.5 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.

## 12.9.6 enum dcp\_swap\_t

## 12.10 Function Documentation

### 12.10.1 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. |

### 12.10.2 void DCP\_Deinit ( **DCP\_Type** \* *base* )

Reset DCP and Disable DCP clock.

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | DCP base address |
|-------------|------------------|

### 12.10.3 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->enableChannnel = kDCP_chEnableAll;` `dcpConfig->enableChannelInterrupt = kDCP_chIntDisable;`

Parameters

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

#### 12.10.4 **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.

## 12.11 DCP AES blocking driver

### 12.11.1 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.*

### 12.11.2 Function Documentation

#### 12.11.2.1 `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

### **12.11.2.2 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                                           |
| <i>out</i> | <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

### **12.11.2.3 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

#### **12.11.2.4 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

#### **12.11.2.5 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

## 12.12 DCP AES non-blocking driver

### 12.12.1 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.*

### 12.12.2 Function Documentation

#### 12.12.2.1 `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.

**12.12.2.2 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.

#### 12.12.2.3 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.

#### 12.12.2.4 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.

## 12.13 DCP HASH driver

### 12.13.1 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.*

## 12.13.2 Data Structure Documentation

### 12.13.2.1 struct dcp\_hash\_ctx\_t

## 12.13.3 Macro Definition Documentation

### 12.13.3.1 #define DCP\_SHA\_BLOCK\_SIZE 128U

internal buffer block size

### 12.13.3.2 #define DCP\_HASH\_CTX\_SIZE 64

## 12.13.4 Enumeration Type Documentation

### 12.13.4.1 enum dcp\_hash\_algo\_t

Enumerator

*kDCP\_Sha1* SHA\_1.  
*kDCP\_Sha256* SHA\_256.  
*kDCP\_Crc32* CRC\_32.

## 12.13.5 Function Documentation

### 12.13.5.1 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.        |
| <i>out</i> | <i>ctx</i>    | Output hash context                                |
|            | <i>algo</i>   | Underlaying algorithm to use for hash computation. |

Returns

Status of initialization

### 12.13.5.2 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 function 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

### 12.13.5.3 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 output[] buffer. On function return, it stores the number of updated output bytes. |

Returns

Status of the hash finish operation

### 12.13.5.4 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 13

## DMAMUX: Direct Memory Access Multiplexer Driver

### 13.1 Overview

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

### 13.2 Typical use case

#### 13.2.1 DMAMUX Operation

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

#### Driver version

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

#### DMAMUX Initialization and de-initialization

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

#### DMAMUX Channel Operation

- static void `DMAMUX_EnableChannel` (DMAMUX\_Type \*base, uint32\_t channel)  
*Enables the DMAMUX channel.*
- static void `DMAMUX_DisableChannel` (DMAMUX\_Type \*base, uint32\_t channel)  
*Disables the DMAMUX channel.*
- static void `DMAMUX_SetSource` (DMAMUX\_Type \*base, uint32\_t channel, uint32\_t source)  
*Configures the DMAMUX channel source.*
- static void `DMAMUX_EnablePeriodTrigger` (DMAMUX\_Type \*base, uint32\_t channel)  
*Enables the DMAMUX period trigger.*
- static void `DMAMUX_DisablePeriodTrigger` (DMAMUX\_Type \*base, uint32\_t channel)  
*Disables the DMAMUX period trigger.*
- static void `DMAMUX_EnableAlwaysOn` (DMAMUX\_Type \*base, uint32\_t channel, bool enable)  
*Enables the DMA channel to be always ON.*

### 13.3 Macro Definition Documentation

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

## 13.4 Function Documentation

### 13.4.1 void DMAMUX\_Init ( **DMAMUX\_Type** \* *base* )

This function ungates the DMAMUX clock.

Parameters

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

### 13.4.2 void DMAMUX\_Deinit ( DMAMUX\_Type \* *base* )

This function gates the DMAMUX clock.

Parameters

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

### 13.4.3 static void DMAMUX\_EnableChannel ( DMAMUX\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This function enables the DMAMUX channel.

Parameters

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

### 13.4.4 static void DMAMUX\_DisableChannel ( DMAMUX\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This function disables the DMAMUX channel.

Note

The user must disable the DMAMUX channel before configuring it.

Parameters

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

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

### 13.4.5 static void DMAMUX\_SetSource ( DMAMUX\_Type \* *base*, uint32\_t *channel*, uint32\_t *source* ) [inline], [static]

Parameters

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

### 13.4.6 static void DMAMUX\_EnablePeriodTrigger ( DMAMUX\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This function enables the DMAMUX period trigger feature.

Parameters

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

### 13.4.7 static void DMAMUX\_DisablePeriodTrigger ( DMAMUX\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This function disables the DMAMUX period trigger.

Parameters

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

### 13.4.8 static void DMAMUX\_EnableAlwaysOn ( DMAMUX\_Type \* *base*, uint32\_t *channel*, bool *enable* ) [inline], [static]

This function enables the DMAMUX channel always ON feature.

## Parameters

|                |                                                                                  |
|----------------|----------------------------------------------------------------------------------|
| <i>base</i>    | DMAMUX peripheral base address.                                                  |
| <i>channel</i> | DMAMUX channel number.                                                           |
| <i>enable</i>  | Switcher of the always ON feature. "true" means enabled, "false" means disabled. |

# Chapter 14

## eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

### 14.1 Overview

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

### 14.2 Typical use case

#### 14.2.1 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** type, uint32\_t linkedChannel)  
*Sets the channel link for the eDMA transfer.*
- void **EDMA\_SetBandWidth** (DMA\_Type \*base, uint32\_t channel, **edma\_bandwidth\_t** bandWidth)  
*Sets the bandwidth for the eDMA transfer.*
- void **EDMA\_SetModulo** (DMA\_Type \*base, uint32\_t channel, **edma\_modulo\_t** srcModulo, **edma\_modulo\_t** destModulo)  
*Sets the source modulo and the destination modulo for the eDMA transfer.*
- static void **EDMA\_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 type, uint32\_t linkedChannel)  
*Sets the channel link for the eDMA TCD.*
- static void [EDMA\\_TcdSetBandWidth](#) (edma\_tcd\_t \*tcd, edma\_bandwidth\_t bandWidth)  
*Sets the bandwidth for the eDMA TCD.*
- void [EDMA\\_TcdSetModulo](#) (edma\_tcd\_t \*tcd, edma\_modulo\_t srcModulo, edma\_modulo\_t destModulo)  
*Sets the source modulo and the destination modulo for the eDMA TCD.*
- static void [EDMA\\_TcdEnableAutoStopRequest](#) (edma\_tcd\_t \*tcd, bool enable)  
*Sets the auto stop request for the eDMA TCD.*
- void [EDMA\\_TcdEnableInterrupts](#) (edma\_tcd\_t \*tcd, uint32\_t mask)  
*Enables the interrupt source for the eDMA TCD.*
- void [EDMA\\_TcdDisableInterrupts](#) (edma\_tcd\_t \*tcd, uint32\_t mask)  
*Disables the interrupt source for the eDMA TCD.*
- void [EDMA\\_TcdSetMajorOffsetConfig](#) (edma\_tcd\_t \*tcd, int32\_t sourceOffset, int32\_t destOffset)  
*Configures the eDMA TCD major offset feature.*

## eDMA Channel Transfer Operation

- static void [EDMA\\_EnableChannelRequest](#) (DMA\_Type \*base, uint32\_t channel)  
*Enables the eDMA hardware channel request.*
- static void [EDMA\\_DisableChannelRequest](#) (DMA\_Type \*base, uint32\_t channel)  
*Disables the eDMA hardware channel request.*
- static void [EDMA\\_TriggerChannelStart](#) (DMA\_Type \*base, uint32\_t channel)  
*Starts the eDMA transfer by using the software trigger.*

## eDMA Channel Status Operation

- uint32\_t [EDMA\\_GetRemainingMajorLoopCount](#) (DMA\_Type \*base, uint32\_t channel)  
*Gets the remaining major loop count from the eDMA current channel TCD.*
- static uint32\_t [EDMA\\_GetErrorStatusFlags](#) (DMA\_Type \*base)  
*Gets the eDMA channel error status flags.*
- uint32\_t [EDMA\\_GetChannelStatusFlags](#) (DMA\_Type \*base, uint32\_t channel)  
*Gets the eDMA channel status flags.*
- void [EDMA\\_ClearChannelStatusFlags](#) (DMA\_Type \*base, uint32\_t channel, uint32\_t mask)  
*Clears the eDMA channel status flags.*

## eDMA Transactional Operation

- void [EDMA\\_CreateHandle](#) (edma\_handle\_t \*handle, DMA\_Type \*base, uint32\_t channel)  
*Creates the eDMA handle.*

- void **EDMA\_InstallTCDMemory** (**edma\_handle\_t** \*handle, **edma\_tcd\_t** \*tcdPool, **uint32\_t** tcdSize)
 

*Installs the TCDs memory pool into the eDMA handle.*
- void **EDMA\_SetCallback** (**edma\_handle\_t** \*handle, **edma\_callback** callback, void \*userData)
 

*Installs a callback function for the eDMA transfer.*
- void **EDMA\_PrepTransferConfig** (**edma\_transfer\_config\_t** \*config, void \*srcAddr, **uint32\_t** srcWidth, **int16\_t** srcOffset, void \*destAddr, **uint32\_t** destWidth, **int16\_t** destOffset, **uint32\_t** bytesEachRequest, **uint32\_t** transferBytes)
 

*Prepares the eDMA transfer structure configurations.*
- void **EDMA\_PrepTransfer** (**edma\_transfer\_config\_t** \*config, void \*srcAddr, **uint32\_t** srcWidth, void \*destAddr, **uint32\_t** destWidth, **uint32\_t** bytesEachRequest, **uint32\_t** transferBytes, **edma\_transfer\_type\_t** type)
 

*Prepares the eDMA transfer structure.*
- **status\_t EDMA\_SubmitTransfer** (**edma\_handle\_t** \*handle, const **edma\_transfer\_config\_t** \*config)
 

*Submits the eDMA transfer request.*
- void **EDMA\_StartTransfer** (**edma\_handle\_t** \*handle)
 

*eDMA starts transfer.*
- void **EDMA\_StopTransfer** (**edma\_handle\_t** \*handle)
 

*eDMA stops transfer.*
- void **EDMA\_AbortTransfer** (**edma\_handle\_t** \*handle)
 

*eDMA aborts transfer.*
- static **uint32\_t EDMA\_GetUnusedTCDNumber** (**edma\_handle\_t** \*handle)
 

*Get unused TCD slot number.*
- static **uint32\_t EDMA\_GetNextTCDAddress** (**edma\_handle\_t** \*handle)
 

*Get the next tcd address.*
- void **EDMA\_HandleIRQ** (**edma\_handle\_t** \*handle)
 

*eDMA IRQ handler for the current major loop transfer completion.*

## 14.3 Data Structure Documentation

### 14.3.1 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.

**14.3.2 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

### 14.3.3 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.*

### 14.3.4 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

### 14.3.5 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**

## 14.3.6 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`

## 14.4 Macro Definition Documentation

### 14.4.1 `#define FSL_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 3))`

Version 2.4.3.

## 14.5 Typedef Documentation

### 14.5.1 `typedef void(* edma_callback)(struct _edma_handle *handle, void *userData, bool transferDone, uint32_t tclds)`

This callback function is called in the EDMA interrupt handle. In normal mode, run into callback function means the transfer users need is done. In scatter gather mode, run into callback function means a transfer control block (tcd) is finished. Not all transfer finished, users can get the finished tcd numbers using interface EDMA\_GetUnusedTCDNumber.

Parameters

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>       | EDMA handle pointer, users shall not touch the values inside.                                                                                                                                                                                                                                                                                                                                                                                    |
| <i>userData</i>     | The callback user parameter pointer. Users can use this parameter to involve things users need to change in EDMA callback function.                                                                                                                                                                                                                                                                                                              |
| <i>transferDone</i> | If the current loaded transfer done. In normal mode it means if all transfer done. In scatter gather mode, this parameter shows is the current transfer block in EDM-A register is done. As the load of core is different, it will be different if the new tcd loaded into EDMA registers while this callback called. If true, it always means new tcd still not loaded into registers, while false means new tcd already loaded into registers. |
| <i>tcds</i>         | How many tcds are done from the last callback. This parameter only used in scatter gather mode. It tells user how many tcds are finished between the last callback and this.                                                                                                                                                                                                                                                                     |

## 14.6 Enumeration Type Documentation

### 14.6.1 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.

### 14.6.2 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.

#### 14.6.3 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.

#### 14.6.4 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.

#### 14.6.5 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

#### 14.6.6 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.

#### 14.6.7 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.

#### 14.6.8 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.

#### 14.6.9 anonymous enum

Enumerator

- kStatus\_EDMA\_QueueFull* TCD queue is full.
- kStatus\_EDMA\_Busy* Channel is busy and can't handle the transfer request.

## 14.7 Function Documentation

### 14.7.1 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.

#### 14.7.2 void EDMA\_Deinit ( DMA\_Type \* *base* )

This function gates the eDMA clock.

Parameters

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

#### 14.7.3 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.       |

#### 14.7.4 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. |
|---------------|------------------------------------------------|

#### 14.7.5 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. |

#### 14.7.6 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. |

#### 14.7.7 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.

#### 14.7.8 void EDMA\_SetTransferConfig ( DMA\_Type \* *base*, uint32\_t *channel*, const edma\_transfer\_config\_t \* *config*, edma\_tcd\_t \* *nextTcd* )

This function configures the transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the TCD address. Example:

```
* edma_transfer_t config;
* edma_tcd_t tcd;
* config.srcAddr = ...;
* config.destAddr = ...;
* ...
* EDMA_SetTransferConfig(DMA0, channel, &config, &tcd);
*
```

## Parameters

|                |                                                                                               |
|----------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                                                                 |
| <i>channel</i> | eDMA channel number.                                                                          |
| <i>config</i>  | Pointer to eDMA transfer configuration structure.                                             |
| <i>nextTcd</i> | Point to TCD structure. It can be NULL if users do not want to enable scatter/gather feature. |

## Note

If nextTcd is not NULL, it means scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the eDMA\_ResetChannel.

#### 14.7.9 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. |

#### **14.7.10 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. |

#### **14.7.11 void EDMA\_SetChannelLink ( DMA\_Type \* *base*, uint32\_t *channel*, edma\_channel\_link\_type\_t *type*, uint32\_t *linkedChannel* )**

This function configures either the minor link or the major link mode. The minor link means that the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Parameters

|                |                                                                                                                                                                              |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                                                                                                                                                |
| <i>channel</i> | eDMA channel number.                                                                                                                                                         |
| <i>type</i>    | A channel link type, which can be one of the following: <ul style="list-style-type: none"><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.

#### 14.7.12 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> |

#### 14.7.13 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. |

**14.7.14 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). |

**14.7.15 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). |

**14.7.16 void EDMA\_EnableChannelInterrupts ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *mask* )**

Parameters

|                |                                                                                                     |
|----------------|-----------------------------------------------------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                                                                       |
| <i>channel</i> | eDMA channel number.                                                                                |
| <i>mask</i>    | The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type. |

**14.7.17 void EDMA\_DisableChannelInterrupts ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *mask* )**

Parameters

|                |                                                                                           |
|----------------|-------------------------------------------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                                                             |
| <i>channel</i> | eDMA channel number.                                                                      |
| <i>mask</i>    | The mask of the interrupt source to be set. Use the defined edma_interrupt_enable_t type. |

#### 14.7.18 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. |

#### 14.7.19 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.

#### 14.7.20 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 STCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```

* edma_transfer_t config = {
* ...
* }
* edma_tcd_t tcd __aligned(32);
* edma_tcd_t nextTcd __aligned(32);
* EDMA_TcdSetTransferConfig(&tcd, &config, &nextTcd);
*

```

## Parameters

|                |                                                                                                          |
|----------------|----------------------------------------------------------------------------------------------------------|
| <i>tcd</i>     | Pointer to the TCD structure.                                                                            |
| <i>config</i>  | Pointer to eDMA transfer configuration structure.                                                        |
| <i>nextTcd</i> | Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature. |

## Note

TCD address should be 32 bytes aligned or it causes an eDMA error.

If the nextTcd is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA\_TcdReset.

#### 14.7.21 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. |

#### 14.7.22 void EDMA\_TcdSetChannelLink ( *edma\_tcd\_t \* tcd, edma\_channel\_link\_type\_t type, uint32\_t linkedChannel* )

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

## Note

Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

|                      |                                                                                                                                                           |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>tcd</i>           | Point to the TCD structure.                                                                                                                               |
| <i>type</i>          | Channel link type, it can be one of: <ul style="list-style-type: none"><li>• kEDMA_LinkNone</li><li>• kEDMA_MinorLink</li><li>• kEDMA_MajorLink</li></ul> |
| <i>linkedChannel</i> | The linked channel number.                                                                                                                                |

#### 14.7.23 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> |

#### 14.7.24 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. |

#### 14.7.25 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). |

#### 14.7.26 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. |

#### 14.7.27 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. |

#### 14.7.28 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. |

#### **14.7.29 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.          |

#### **14.7.30 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.          |

#### **14.7.31 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.          |

**14.7.32 `uint32_t EDMA_GetRemainingMajorLoopCount ( DMA_Type * base,`**  
**`uint32_t channel )`**

This function checks the TCD (Task Control Descriptor) status for a specified eDMA channel and returns the number of major loop count that has not finished.

## Parameters

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

## Returns

Major loop count which has not been transferred yet for the current TCD.

## Note

1. This function can only be used to get unfinished major loop count of transfer without the next TCD, or it might be inaccuracy.
  1. The unfinished/remaining transfer bytes cannot be obtained directly from registers while the channel is running. Because to calculate the remaining bytes, the initial NBYTES configured in DMA\_TCDn\_NBYTES\_MLNO register is needed while the eDMA IP does not support getting it while a channel is active. In another word, the NBYTES value reading is always the actual (decrementing) NBYTES value the dma\_engine is working with while a channel is running. Consequently, to get the remaining transfer bytes, a software-saved initial value of NBYTES (for example copied before enabling the channel) is needed. The formula to calculate it is shown below: RemainingBytes = RemainingMajorLoopCount \* NBYTE-S(initially configured)

#### 14.7.33 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.

#### 14.7.34 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.

#### 14.7.35 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. |

#### 14.7.36 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.                                                          |

#### 14.7.37 void EDMA\_InstallTCDMemory ( edma\_handle\_t \* *handle*, edma\_tcd\_t \* *tcdPool*, uint32\_t *tcdSize* )

This function is called after the EDMA\_CreateHandle to use scatter/gather feature. This function shall only be used while users need to use scatter gather mode. Scatter gather mode enables EDMA to load a new transfer control block (tcd) in hardware, and automatically reconfigure that DMA channel for a

new transfer. Users need to prepare tcd memory and also configure tcds using interface EDMA\_SubmitTransfer.

Parameters

|                |                                                           |
|----------------|-----------------------------------------------------------|
| <i>handle</i>  | eDMA handle pointer.                                      |
| <i>tcdPool</i> | A memory pool to store TCDs. It must be 32 bytes aligned. |
| <i>tcdSize</i> | The number of TCD slots.                                  |

#### 14.7.38 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. |

#### 14.7.39 void EDMA\_PrepTransferConfig ( *edma\_transfer\_config\_t \* config, void \* srcAddr, uint32\_t srcWidth, int16\_t srcOffset, void \* destAddr, uint32\_t destWidth, int16\_t destOffset, uint32\_t bytesEachRequest, uint32\_t transferBytes* )

This function prepares the transfer configuration structure according to the user input.

Parameters

|                         |                                                                   |
|-------------------------|-------------------------------------------------------------------|
| <i>config</i>           | The user configuration structure of type <i>edma_transfer_t</i> . |
| <i>srcAddr</i>          | eDMA transfer source address.                                     |
| <i>srcWidth</i>         | eDMA transfer source address width(bytes).                        |
| <i>srcOffset</i>        | source address offset.                                            |
| <i>destAddr</i>         | eDMA transfer destination address.                                |
| <i>destWidth</i>        | eDMA transfer destination address width(bytes).                   |
| <i>destOffset</i>       | destination address offset.                                       |
| <i>bytesEachRequest</i> | eDMA transfer bytes per channel request.                          |
| <i>transferBytes</i>    | eDMA transfer bytes to be transferred.                            |

## Note

The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

#### **14.7.40 void EDMA\_PrepTransfer ( *edma\_transfer\_config\_t \* config, void \* srcAddr, uint32\_t srcWidth, void \* destAddr, uint32\_t destWidth, uint32\_t bytesEachRequest, uint32\_t transferBytes, edma\_transfer\_type\_t type* )**

This function prepares the transfer configuration structure according to the user input.

## Parameters

|                         |                                                                         |
|-------------------------|-------------------------------------------------------------------------|
| <i>config</i>           | The user configuration structure of type <code>edma_transfer_t</code> . |
| <i>srcAddr</i>          | eDMA transfer source address.                                           |
| <i>srcWidth</i>         | eDMA transfer source address width(bytes).                              |
| <i>destAddr</i>         | eDMA transfer destination address.                                      |
| <i>destWidth</i>        | eDMA transfer destination address width(bytes).                         |
| <i>bytesEachRequest</i> | eDMA transfer bytes per channel request.                                |
| <i>transferBytes</i>    | eDMA transfer bytes to be transferred.                                  |
| <i>type</i>             | eDMA transfer type.                                                     |

## Note

The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

#### **14.7.41 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.   |

#### 14.7.42 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. |
|---------------|----------------------|

#### 14.7.43 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. |
|---------------|----------------------|

#### 14.7.44 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. |
|---------------|---------------------|

#### **14.7.45 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.

#### **14.7.46 static uint32\_t EDMA\_GetNextTCDAccount ( edma\_handle\_t \* *handle* ) [inline], [static]**

This function gets the next tcd address. If this is last TCD, return 0.

Parameters

|               |                     |
|---------------|---------------------|
| <i>handle</i> | DMA handle pointer. |
|---------------|---------------------|

Returns

The next TCD address.

#### **14.7.47 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 15

## EWM: External Watchdog Monitor Driver

### 15.1 Overview

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

### 15.2 Typical use case

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

### Data Structures

- struct `ewm_config_t`  
*Data structure for EWM configuration. [More...](#)*

### Enumerations

- enum `ewm_lpo_clock_source_t` {  
    `kEWM_LpoClockSource0` = 0U,  
    `kEWM_LpoClockSource1` = 1U,  
    `kEWM_LpoClockSource2` = 2U,  
    `kEWM_LpoClockSource3` = 3U }  
*Describes EWM clock source.*
- 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.*

## 15.3 Data Structure Documentation

### 15.3.1 struct ewm\_config\_t

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.*
- [ewm\\_lpo\\_clock\\_source\\_t clockSource](#)  
*Clock source select.*
- uint8\_t [prescaler](#)  
*Clock prescaler value.*
- uint8\_t [compareLowValue](#)  
*Compare low-register value.*
- uint8\_t [compareHighValue](#)  
*Compare high-register value.*

## 15.4 Macro Definition Documentation

### 15.4.1 #define FSL\_EWM\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 3))

## 15.5 Enumeration Type Documentation

### 15.5.1 enum ewm\_lpo\_clock\_source\_t

Enumerator

- kEWM\_LpoClockSource0** EWM clock sourced from lpo\_clk[0].
- kEWM\_LpoClockSource1** EWM clock sourced from lpo\_clk[1].
- kEWM\_LpoClockSource2** EWM clock sourced from lpo\_clk[2].

***kEWM\_LpoClockSource3*** EWM clock sourced from lpo\_clk[3].

### 15.5.2 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.

### 15.5.3 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.

## 15.6 Function Documentation

### 15.6.1 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 |

### 15.6.2 void EWM\_Deinit ( EWM\_Type \* *base* )

This function is used to shut down the EWM.

Parameters

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

### 15.6.3 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](#)

### 15.6.4 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> |

### 15.6.5 static void EWM\_DisableInterrupts ( *EWM\_Type* \* *base*, *uint32\_t* *mask* ) [inline], [static]

This function disables 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> |

### 15.6.6 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.

### 15.6.7 void EWM\_Refresh ( EWM\_Type \* *base* )

This function resets the EWM counter to zero.

## Parameters

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

# Chapter 16

## FlexIO: FlexIO Driver

### 16.1 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](#)

## 16.2 FlexIO Driver

### 16.2.1 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` = 0x1U,
   
  `kFLEXIO_TimerDecSrcOnPinInputShiftPinInput` = 0x2U,
   
  `kFLEXIO_TimerDecSrcOnTriggerInputShiftTriggerInput` = 0x3U }
   
    *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_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,
   
kFLEXIO\_ShifterModeState = 0x6U,
   
kFLEXIO\_ShifterModeLogic = 0x7U }

*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,
   
kFLEXIO\_ShifterBufferNibbleByteSwapped = 0x4U,
   
kFLEXIO\_ShifterBufferHalfWordSwapped = 0x5U,
   
kFLEXIO\_ShifterBufferNibbleSwapped = 0x6U }

*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, 0, 4))`  
*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 FLEXIOGetInstance (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.*
- `static uint32_t FLEXIO_ReadPinInput (FLEXIO_Type *base)`  
*Reads the input data on each of the FlexIO pins.*
- `static uint8_t FLEXIO_GetShifterState (FLEXIO_Type *base)`  
*Gets the current state pointer for state mode use.*
- `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.*

## 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)`

- static void [FLEXIO\\_EnableShifterErrorInterrupts](#) (FLEXIO\_Type \*base, uint32\_t mask)  
*Disables the shifter status interrupt.*
- static void [FLEXIO\\_DisableShifterErrorInterrupts](#) (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.*

### 16.2.2 Data Structure Documentation

#### 16.2.2.1 struct flexio\_config\_t

##### Data Fields

- bool [enableFlexio](#)  
*Enable/disable FlexIO module.*
- bool [enableInDoze](#)

- **bool enableInDebug**  
Enable/disable FlexIO operation in doze mode.
- **bool enableFastAccess**  
Enable/disable FlexIO operation in debug mode.  
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**

#### 16.2.2.2 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`

### 16.2.2.3 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.*
- `uint32_t parallelWidth`  
*Configures the parallel width when using parallel mode.*

- `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) `uint32_t flexio_shifter_config_t::parallelWidth`
- (8) `flexio_shifter_input_source_t flexio_shifter_config_t::inputSource`
- (9) `flexio_shifter_stop_bit_t flexio_shifter_config_t::shifterStop`
- (10) `flexio_shifter_start_bit_t flexio_shifter_config_t::shifterStart`

## 16.2.3 Macro Definition Documentation

16.2.3.1 `#define FSL_FLEXIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))`

16.2.3.2 `#define FLEXIO_TIMER_TRIGGER_SEL_PININPUT( x ) ((uint32_t)(x) << 1U)`

## 16.2.4 Typedef Documentation

16.2.4.1 `typedef void(* flexio_isr_t)(void *base, void *handle)`

## 16.2.5 Enumeration Type Documentation

### 16.2.5.1 enum flexio\_timer\_trigger\_polarity\_t

Enumerator

`kFLEXIO_TimerTriggerPolarityActiveHigh` Active high.

`kFLEXIO_TimerTriggerPolarityActiveLow` Active low.

### 16.2.5.2 enum flexio\_timer\_trigger\_source\_t

Enumerator

*kFLEXIO\_TimerTriggerSourceExternal* External trigger selected.

*kFLEXIO\_TimerTriggerSourceInternal* Internal trigger selected.

### 16.2.5.3 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.

### 16.2.5.4 enum flexio\_pin\_polarity\_t

Enumerator

*kFLEXIO\_PinActiveHigh* Active high.

*kFLEXIO\_PinActiveLow* Active low.

### 16.2.5.5 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.

### 16.2.5.6 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.

### 16.2.5.7 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.

### 16.2.5.8 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.

### 16.2.5.9 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.

### 16.2.5.10 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.

### 16.2.5.11 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.

### 16.2.5.12 enum flexio\_timer\_start\_bit\_condition\_t

Enumerator

***kFLEXIO\_TimerStartBitDisabled*** Start bit disabled.

***kFLEXIO\_TimerStartBitEnabled*** Start bit enabled.

### 16.2.5.13 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.

### 16.2.5.14 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.

***kFLEXIO\_ShifterModeState*** SHIFTBUF contents are used for storing programmable state attributes.

***kFLEXIO\_ShifterModeLogic*** SHIFTBUF contents are used for implementing programmable logic look up table.

### 16.2.5.15 enum flexio\_shifter\_input\_source\_t

Enumerator

***kFLEXIO\_ShifterInputFromPin*** Shifter input from pin.

***kFLEXIO\_ShifterInputFromNextShifterOutput*** Shifter input from Shifter N+1.

### 16.2.5.16 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.

### 16.2.5.17 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.

### 16.2.5.18 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.

***kFLEXIO\_ShifterBufferNibbleByteSwapped*** Shifter Buffer N Nibble Byte Swapped Register.

***kFLEXIO\_ShifterBufferHalfWordSwapped*** Shifter Buffer N Half Word Swapped Register.

***kFLEXIO\_ShifterBufferNibbleSwapped*** Shifter Buffer N Nibble Swapped Register.

## 16.2.6 Function Documentation

### 16.2.6.1 void FLEXIO\_GetDefaultConfig ( flexio\_config\_t \* *userConfig* )

The configuration can used directly to call the FLEXIO\_Configure().

Example:

```
flexio_config_t config;
FLEXIO_GetDefaultConfig(&config);
```

Parameters

|                   |                                                   |
|-------------------|---------------------------------------------------|
| <i>userConfig</i> | pointer to <code>flexio_config_t</code> structure |
|-------------------|---------------------------------------------------|

### 16.2.6.2 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 <code>flexio_config_t</code> structure |

### 16.2.6.3 void FLEXIO\_Deinit ( FLEXIO\_Type \* *base* )

Call this API to stop the FlexIO clock.

Note

After calling this API, call the FLEXO\_Init to use the FlexIO module.

Parameters

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

#### 16.2.6.4 `uint32_t FLEXIO_GetInstance ( FLEXIO_Type * base )`

Parameters

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

#### 16.2.6.5 `void FLEXIO_Reset ( FLEXIO_Type * base )`

Parameters

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

#### 16.2.6.6 `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. |

#### 16.2.6.7 `static uint32_t FLEXIO_ReadPinInput ( FLEXIO_Type * base ) [inline], [static]`

Parameters

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

Returns

FlexIO pin input data

#### 16.2.6.8 `static uint8_t FLEXIO_GetShifterState ( FLEXIO_Type * base ) [inline], [static]`

Parameters

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

Returns

current State pointer

#### 16.2.6.9 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 |

#### 16.2.6.10 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 <code>flexio_timer_config_t</code> structure |

#### 16.2.6.11 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  $((1 \ll \text{shifter index}0) | (1 \ll \text{shifter index}1))$

#### 16.2.6.12 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 ( $1 << \text{shifter index}$ ) |

Note

For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using  $((1 << \text{shifter index}0) | (1 << \text{shifter index}1))$

#### 16.2.6.13 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 ( $1 << \text{shifter index}$ ) |

Note

For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using  $((1 << \text{shifter index}0) | (1 << \text{shifter index}1))$

#### 16.2.6.14 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 ( $1 << \text{shifter index}$ ) |

Note

For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using  $((1 << \text{shifter index}0) | (1 << \text{shifter index}1))$

**16.2.6.15 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 ( $1 << \text{timer index}$ ) |

Note

For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using  $((1 << \text{timer index}0) | (1 << \text{timer index}1))$

#### 16.2.6.16 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 ( $1 << \text{timer index}$ ) |

Note

For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using  $((1 << \text{timer index}0) | (1 << \text{timer index}1))$

#### 16.2.6.17 static uint32\_t FLEXIO\_GetShifterStatusFlags ( FLEXIO\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

Shifter status flags

#### 16.2.6.18 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 ( $1 << \text{shifter index}$ ) |

Note

For clearing multiple shifter status flags, for example, two shifter status flags, can calculate the mask by using  $((1 << \text{shifter index}0) | (1 << \text{shifter index}1))$

#### 16.2.6.19 static uint32\_t FLEXIO\_GetShifterErrorFlags ( FLEXIO\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

Shifter error flags

#### 16.2.6.20 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 ( $1 << \text{shifter index}$ ) |

Note

For clearing multiple shifter error flags, for example, two shifter error flags, can calculate the mask by using  $((1 << \text{shifter index}0) | (1 << \text{shifter index}1))$

#### 16.2.6.21 static uint32\_t FLEXIO\_GetTimerStatusFlags ( FLEXIO\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

Timer status flags

#### **16.2.6.22 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 $(1 << \text{timer index})$ |

Note

For clearing multiple timer status flags, for example, two timer status flags, can calculate the mask by using  $((1 << \text{timer index}0) | (1 << \text{timer index}1))$

#### **16.2.6.23 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  $((1 << \text{shifter index}0) | (1 << \text{shifter index}1))$

Parameters

|             |                                                                                  |
|-------------|----------------------------------------------------------------------------------|
| <i>base</i> | FlexIO peripheral base address                                                   |
| <i>mask</i> | The shifter status mask which can be calculated by $(1 << \text{shifter index})$ |

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

### 16.2.6.24 **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

### 16.2.6.25 **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. |

### 16.2.6.26 **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. |

## 16.2.7 Variable Documentation

### 16.2.7.1 **FLEXIO\_Type\* const s\_flexioBases[]**

### 16.2.7.2 **const clock\_ip\_name\_t s\_flexioClocks[]**

## 16.3 FlexIO Camera Driver

### 16.3.1 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`.

### 16.3.2 Typical use case

#### 16.3.2.1 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)ul6CameraFrameBuffer;
cameraTransfer.dataNum = sizeof(ul6CameraFrameBuffer);
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.*

### 16.3.3 Data Structure Documentation

#### 16.3.3.1 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**

#### 16.3.3.2 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`

### 16.3.3.3 struct flexio\_camera\_transfer\_t

#### Data Fields

- `uint32_t dataAddress`  
*Transfer buffer.*
- `uint32_t dataNum`  
*Transfer num.*

### 16.3.4 Macro Definition Documentation

#### 16.3.4.1 #define FSL\_FLEXIO\_CAMERA\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 3))

#### 16.3.4.2 #define FLEXIO\_CAMERA\_PARALLEL\_DATA\_WIDTH (8U)

### 16.3.5 Enumeration Type Documentation

#### 16.3.5.1 anonymous enum

Enumerator

- `kStatus_FLEXIO_CAMERA_RxBusy` Receiver is busy.  
`kStatus_FLEXIO_CAMERA_RxIdle` Camera receiver is idle.

#### 16.3.5.2 enum \_flexio\_camera\_status\_flags

Enumerator

- `kFLEXIO_CAMERA_RxDataRegFullFlag` Receive buffer full flag.  
`kFLEXIO_CAMERA_RxErrorFlag` Receive buffer error flag.

### 16.3.6 Function Documentation

#### 16.3.6.1 void FLEXIO\_CAMERA\_Init ( **FLEXIO\_CAMERA\_Type** \* *base*, const **flexio\_camera\_config\_t** \* *config* )

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | Pointer to <b>FLEXIO_CAMERA_Type</b> structure     |
| <i>config</i> | Pointer to <b>flexio_camera_config_t</b> structure |

#### 16.3.6.2 void FLEXIO\_CAMERA\_Deinit ( **FLEXIO\_CAMERA\_Type** \* *base* )

Note

After calling this API, call **FLEXO\_CAMERA\_Init** to use the FlexIO Camera module.

Parameters

|             |                                                |
|-------------|------------------------------------------------|
| <i>base</i> | Pointer to <b>FLEXIO_CAMERA_Type</b> structure |
|-------------|------------------------------------------------|

#### 16.3.6.3 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 <b>flexio_camera_config_t</b> structure |
|---------------|--------------------------------------------------------|

#### 16.3.6.4 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.   |

**16.3.6.5 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

**16.3.6.6 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>• <a href="#">kFLEXIO_CAMERA_RxDataRegFullFlag</a></li> <li>• <a href="#">kFLEXIO_CAMERA_RxErrorFlag</a></li> </ul> |

**16.3.6.7 void FLEXIO\_CAMERA\_EnableInterrupt ( [FLEXIO\\_CAMERA\\_Type](#) \* *base* )**

Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | Pointer to the device. |
|-------------|------------------------|

**16.3.6.8 void FLEXIO\_CAMERA\_DisableInterrupt ( [FLEXIO\\_CAMERA\\_Type](#) \* *base* )**

Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | Pointer to the device. |
|-------------|------------------------|

### 16.3.6.9 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.

### 16.3.6.10 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 .

## 16.3.7 FlexIO eDMA Camera Driver

### 16.3.7.1 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.*

### 16.3.7.2 Data Structure Documentation

#### 16.3.7.2.1 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`

## 16.3.7.3 Macro Definition Documentation

16.3.7.3.1 `#define FSL_FLEXIO_CAMERA_EDMA_DRIVER_VERSION(MAKE_VERSION(2, 1, 3))`

## 16.3.7.4 Typedef Documentation

16.3.7.4.1 `typedef void(* flexio_camera_edma_transfer_callback_t)(FLEXIO_CAMERA_Type  
*base, flexio_camera_edma_handle_t *handle, status_t status, void *userData)`

## 16.3.7.5 Function Documentation

16.3.7.5.1 `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

|                   |                                                     |
|-------------------|-----------------------------------------------------|
| <code>base</code> | 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. |

#### 16.3.7.5.2 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.  |

#### 16.3.7.5.3 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. |
|---------------|-------------------------------------------------------|

**16.3.7.5.4 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 <b>FLEXIO_CAMERA_Type</b> .                   |
| <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. |

## 16.4 FlexIO I2C Master Driver

### 16.4.1 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.

### 16.4.2 Typical use case

#### 16.4.2.1 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, 4, 0))

Initialization and deinitialization

- status_t FLEXIO_I2C_CheckForBusyBus (FLEXIO_I2C_Type *base)
 Make sure the bus isn't already pulled down.
- status_t FLEXIO_I2C_MasterInit (FLEXIO_I2C_Type *base, flexio_i2c_master_config_t *masterConfig, uint32_t srcClock_Hz)
 Ungates 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.

16.4.3 Data Structure Documentation

16.4.3.1 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**

16.4.3.2 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`

16.4.3.3 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`

16.4.3.4 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`

16.4.4 Macro Definition Documentation

- 16.4.4.1 `#define I2C_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`

16.4.5 Typedef Documentation

16.4.5.1 `typedef void(* flexio_i2c_master_transfer_callback_t)(FLEXIO_I2C_Type *base, flexio_i2c_master_handle_t *handle, status_t status, void *userData)`

16.4.6 Enumeration Type Documentation

16.4.6.1 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.

16.4.6.2 `enum _flexio_i2c_master_interrupt`

Enumerator

kFLEXIO_I2C_TxEmptyInterruptEnable Tx buffer empty interrupt enable.

kFLEXIO_I2C_RxFullInterruptEnable Rx buffer full interrupt enable.

16.4.6.3 `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.

16.4.6.4 `enum flexio_i2c_direction_t`

Enumerator

kFLEXIO_I2C_Write Master send to slave.

kFLEXIO_I2C_Read Master receive from slave.

16.4.7 Function Documentation

16.4.7.1 `status_t FLEXIO_I2C_CheckForBusyBus (FLEXIO_I2C_Type * base)`

Check the FLEXIO pin status to see whether either of SDA and SCL pin is pulled down.

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure..
-------------	--

Return values

<i>kStatus_Success</i>	
<i>kStatus_FLEXIO_I2C_Busy</i>	

16.4.7.2 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

16.4.7.3 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.
-------------	---

16.4.7.4 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.
---------------------	--

16.4.7.5 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.

16.4.7.6 [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.

16.4.7.7 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

16.4.7.8 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

16.4.7.9 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.

16.4.7.10 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
--------------------	--------------------

16.4.7.11 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

16.4.7.12 void FLEXIO_I2C_MasterStop (FLEXIO_I2C_Type * *base*)

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
-------------	---

16.4.7.13 void FLEXIO_I2C_MasterRepeatedStart (FLEXIO_I2C_Type * *base*)

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
-------------	---

16.4.7.14 void FLEXIO_I2C_MasterAbortStop (FLEXIO_I2C_Type * *base*)

Parameters

<i>base</i>	Pointer to FLEXIO_I2C_Type structure.
-------------	---

16.4.7.15 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.

16.4.7.16 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.

16.4.7.17 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.

16.4.7.18 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.

16.4.7.19 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.

16.4.7.20 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.

16.4.7.21 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`.

16.4.7.22 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 flexio_i2c_master_handle_t 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.

16.4.7.23 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 flexio_i2c_master_handle_t 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.

16.4.7.24 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_NoTransferIn-Progress</i>	There is not a non-blocking transaction currently in progress.
<i>kStatus_Success</i>	Successfully return the count.

16.4.7.25 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

16.4.7.26 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

16.5 FlexIO I2S Driver

16.5.1 Overview

The MCUXpresso SDK provides a peripheral driver for I2S function using Flexible I/O module of MCUXpresso 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.

16.5.2 Typical use case

16.5.2.1 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)
{
}

// ...
}

```

16.5.2.2 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 I2S interrupt mask.
- enum {

kFLEXIO_I2S_TxDataRegEmptyFlag = 0x1U,

kFLEXIO_I2S_RxDataRegFullFlag = 0x2U }

_flexio_i2s_status_flags Define 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.

16.5.3 Data Structure Documentation

16.5.3.1 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.

16.5.3.2 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.

16.5.3.3 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.

16.5.3.4 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`

16.5.3.5 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.

16.5.4 Macro Definition Documentation

16.5.4.1 `#define FSL_FLEXIO_I2S_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))`

16.5.4.2 `#define I2S_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`

16.5.4.3 `#define FLEXIO_I2S_XFER_QUEUE_SIZE (4U)`

16.5.5 Enumeration Type Documentation

16.5.5.1 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.

16.5.5.2 enum flexio_i2s_master_slave_t

Enumerator

kFLEXIO_I2S_Master Master mode.
kFLEXIO_I2S_Slave Slave mode.

16.5.5.3 anonymous enum

Enumerator

kFLEXIO_I2S_TxDataRegEmptyInterruptEnable Transmit buffer empty interrupt enable.
kFLEXIO_I2S_RxDataRegFullInterruptEnable Receive buffer full interrupt enable.

16.5.5.4 anonymous enum

Enumerator

kFLEXIO_I2S_TxDataRegEmptyFlag Transmit buffer empty flag.
kFLEXIO_I2S_RxDataRegFullFlag Receive buffer full flag.

16.5.5.5 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.

16.5.5.6 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.

16.5.6 Function Documentation

16.5.6.1 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.

16.5.6.2 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
---------------	---

16.5.6.3 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 FLEXO_I2S_Init to use the FlexIO I2S module.

Parameters

<i>base</i>	FlexIO I2S base pointer
-------------	-------------------------

16.5.6.4 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.

16.5.6.5 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.

16.5.6.6 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

16.5.6.7 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

16.5.6.8 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.

16.5.6.9 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.

16.5.6.10 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.

16.5.6.11 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.

16.5.6.12 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.

16.5.6.13 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.

16.5.6.14 **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.

16.5.6.15 **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.

16.5.6.16 **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.

**16.5.6.17 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.

**16.5.6.18 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.

16.5.6.19 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.

16.5.6.20 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.

16.5.6.21 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.

16.5.6.22 `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.

16.5.6.23 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

16.5.6.24 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

16.5.6.25 **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_NoTransferIn-Progress</i>	There is not a non-blocking transaction currently in progress.

16.5.6.26 **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 received.

Returns

count Bytes received.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferIn-Progress</i>	There is not a non-blocking transaction currently in progress.

16.5.6.27 **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
------------------	--

16.5.6.28 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.

16.5.7 FlexIO eDMA I2S Driver

16.5.7.1 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.

16.5.7.2 Data Structure Documentation

16.5.7.2.1 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**

16.5.7.3 Macro Definition Documentation

16.5.7.3.1 #define FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 7))

16.5.7.4 Function Documentation

16.5.7.4.1 void FLEXIO_I2S_TransferTxCreateHandleEDMA (*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.

16.5.7.4.2 void FLEXIO_I2S_TransferRxCreateHandleEDMA (*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.

16.5.7.4.3 void FLEXIO_I2S_TransferSetFormatEDMA (*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.

16.5.7.4.4 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.

16.5.7.4.5 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.

16.5.7.4.6 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.

16.5.7.4.7 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.

16.5.7.4.8 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.

16.5.7.4.9 status_t FLEXIO_I2S_TransferGetReceiveCountEDMA (***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.

16.6 FlexIO MCU Interface LCD Driver

16.6.1 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.

16.6.2 Typical use case

16.6.2.1 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, 120000000);

// 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);

```

16.6.2.2 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_mculed_config_t**
Define FlexIO MCULCD configuration structure. [More...](#)
- struct **flexio_mculed_transfer_t**
Define FlexIO MCULCD transfer structure. [More...](#)
- struct **flexio_mculed_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_mculed_pin_func_t**)(bool set)
Function to set or clear the CS and RS pin.
- typedef void(* **flexio_mculed_transfer_callback_t**)(FLEXIO_MCULCD_Type *base, flexio_mculed_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_mculed_pixel_format_t** {
 kFLEXIO_MCULCD_RGB565 = 0,
 kFLEXIO_MCULCD_BGR565,
 kFLEXIO_MCULCD_RGB888,
 kFLEXIO_MCULCD_BGR888 }
 Define FlexIO MCULCD pixel format.
- enum **flexio_mculed_bus_t** {
 kFLEXIO_MCULCD_8080,
 kFLEXIO_MCULCD_6800 }
 Define FlexIO MCULCD bus type.
- enum **_flexio_mculed_interrupt_enable** {
 kFLEXIO_MCULCD_TxEmptyInterruptEnable = (1U << 0U),

- `kFLEXIO_MCULCD_RxFullInterruptEnable = (1U << 1U) }`
Define FlexIO MCULCD interrupt mask.
- enum `_flexio_mculed_status_flags {`
`kFLEXIO_MCULCD_TxEmptyFlag = (1U << 0U),`
`kFLEXIO_MCULCD_RxFullFlag = (1U << 1U) }`
Define FlexIO MCULCD status mask.
- enum `_flexio_mculed_dma_enable {`
`kFLEXIO_MCULCD_TxDmaEnable = 0x1U,`
`kFLEXIO_MCULCD_RxDmaEnable = 0x2U }`
Define FlexIO MCULCD DMA mask.
- enum `flexio_mculed_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, 6))`
FlexIO MCULCD driver version.

FlexIO MCULCD Configuration

- `status_t FLEXIO_MCULCD_Init (FLEXIO_MCULCD_Type *base, flexio_mculed_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_mculed_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_mculed_transfer_t *xfer`)

Performs a polling transfer.

Transactional

- `status_t FLEXIO_MCULCD_TransferCreateHandle` (`FLEXIO_MCULCD_Type` *base, `flexio_mculed_handle_t *handle`, `flexio_mculed_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_mculed_handle_t *handle`, `flexio_mculed_transfer_t *xfer`)

Transfer data using IRQ.
- void `FLEXIO_MCULCD_TransferAbort` (`FLEXIO_MCULCD_Type` *base, `flexio_mculed_handle_t *handle`)

Aborts the data transfer, which used IRQ.
- `status_t FLEXIO_MCULCD_TransferGetCount` (`FLEXIO_MCULCD_Type` *base, `flexio_mculed_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.

16.6.3 Data Structure Documentation

16.6.3.1 struct `FLEXIO_MCULCD_Type`

Data Fields

- `FLEXIO_Type * flexioBase`

FlexIO base pointer.
- `flexio_mculed_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_mculed_pin_func_t setCSPin`
Function to set or clear the CS pin.
- `flexio_mculed_pin_func_t setRSPin`
Function to set or clear the RS pin.
- `flexio_mculed_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_mculed_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_mculed_pin_func_t FLEXIO_MCULCD_Type::setCSPin`
- (12) `flexio_mculed_pin_func_t FLEXIO_MCULCD_Type::setRSPin`
- (13) `flexio_mculed_pin_func_t FLEXIO_MCULCD_Type::setRDWRPin`

16.6.3.2 struct flexio_mculed_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 Bps.

Field Documentation

- (1) `bool flexio_mculed_config_t::enable`
- (2) `bool flexio_mculed_config_t::enableInDoze`
- (3) `bool flexio_mculed_config_t::enableInDebug`
- (4) `bool flexio_mculed_config_t::enableFastAccess`
- (5) `uint32_t flexio_mculed_config_t::baudRate_Bps`

16.6.3.3 struct flexio_mculed_transfer_t

Data Fields

- uint32_t `command`
Command to send.
- `flexio_mculed_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_mculed_transfer_t::command`
- (2) `flexio_mculed_transfer_mode_t flexio_mculed_transfer_t::mode`
- (3) `uint32_t flexio_mculed_transfer_t::dataAddrOrSameValue`

When writing or reading array, this is the address of the data array.

(4) `size_t flexio_mculed_transfer_t::dataSize`

16.6.3.4 struct _flexio_mculed_handle

typedef for `flexio_mculed_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_mculed_transfer_callback_t completionCallback`
FlexIO MCULCD transfer completed callback.
- `void *userData`
Callback parameter.

Field Documentation

(1) `uint32_t flexio_mculed_handle_t::dataAddrOrSameValue`

When writing or reading array, this is the address of the data array.

(2) `size_t flexio_mculed_handle_t::dataCount`

(3) `volatile size_t flexio_mculed_handle_t::remainingCount`

(4) `volatile uint32_t flexio_mculed_handle_t::state`

(5) `flexio_mculed_transfer_callback_t flexio_mculed_handle_t::completionCallback`

(6) `void* flexio_mculed_handle_t::userData`

16.6.4 Macro Definition Documentation

16.6.4.1 `#define FSL_FLEXIO_MCULCD_DRIVER_VERSION (MAKE_VERSION(2, 0, 6))`

16.6.4.2 `#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.

16.6.5 Typedef Documentation

16.6.5.1 `typedef void(* flexio_mculed_pin_func_t)(bool set)`

16.6.5.2 `typedef void(* flexio_mculed_transfer_callback_t)(FLEXIO_MCULCD_Type *base, flexio_mculed_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`.

16.6.6 Enumeration Type Documentation

16.6.6.1 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.

16.6.6.2 `enum flexio_mculed_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.

16.6.6.3 `enum flexio_mculed_bus_t`

Enumerator

kFLEXIO_MCULCD_8080 Using Intel 8080 bus.

kFLEXIO_MCULCD_6800 Using Motorola 6800 bus.

16.6.6.4 `enum _flexio_mculed_interrupt_enable`

Enumerator

kFLEXIO_MCULCD_TxEmptyInterruptEnable Transmit buffer empty interrupt enable.

kFLEXIO_MCULCD_RxFullInterruptEnable Receive buffer full interrupt enable.

16.6.6.5 enum _flexio_mculed_status_flags

Enumerator

kFLEXIO_MCULCD_TxEmptyFlag Transmit buffer empty flag.

kFLEXIO_MCULCD_RxFullFlag Receive buffer full flag.

16.6.6.6 enum _flexio_mculed_dma_enable

Enumerator

kFLEXIO_MCULCD_TxDmaEnable Tx DMA request source.

kFLEXIO_MCULCD_RxDmaEnable Rx DMA request source.

16.6.6.7 enum flexio_mculed_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.

16.6.7 Function Documentation

16.6.7.1 status_t FLEXIO_MCULCD_Init (***FLEXIO_MCULCD_Type * base,*** ***flexio_mculed_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 FLEXIO_MCULCD_Type structure.
<i>config</i>	Pointer to the flexio_mculed_config_t 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.

16.6.7.2 void FLEXIO_MCULCD_Deinit (FLEXIO_MCULCD_Type * *base*)

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
-------------	---

16.6.7.3 void FLEXIO_MCULCD_GetDefaultConfig (flexio_mculed_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 flexio_mculed_config_t structure.
---------------	--

16.6.7.4 uint32_t FLEXIO_MCULCD_GetStatusFlags (FLEXIO_MCULCD_Type * *base*)

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
-------------	--

Returns

status flag; OR'ed value or the [_flexio_mculed_status_flags](#).

Note

Don't use this function with DMA APIs.

16.6.7.5 void FLEXIO_MCULCD_ClearStatusFlags (FLEXIO_MCULCD_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>mask</i>	Status to clear, it is the OR'ed value of _flexio_mculcd_status_flags .

Note

Don't use this function with DMA APIs.

16.6.7.6 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 FLEXIO_MCULCD_Type structure.
<i>mask</i>	Interrupts to enable, it is the OR'ed value of _flexio_mculcd_interrupt_enable .

16.6.7.7 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 FLEXIO_MCULCD_Type structure.
<i>mask</i>	Interrupts to disable, it is the OR'ed value of _flexio_mculcd_interrupt_enable .

16.6.7.8 static void FLEXIO_MCULCD_EnableTxDMA (FLEXIO_MCULCD_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
-------------	--

<i>enable</i>	True means enable DMA, false means disable DMA.
---------------	---

16.6.7.9 static void FLEXIO_MCULCD_EnableRxDMA (**FLEXIO_MCULCD_Type** * *base*, **bool** *enable*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>enable</i>	True means enable DMA, false means disable DMA.

16.6.7.10 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 FLEXIO_MCULCD_Type structure.
-------------	---

Returns

FlexIO MCULCD transmit data register address.

16.6.7.11 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 FLEXIO_MCULCD_Type structure.
-------------	---

Returns

FlexIO MCULCD receive data register address.

16.6.7.12 status_t FLEXIO_MCULCD_SetBaudRate (**FLEXIO_MCULCD_Type** * *base*, **uint32_t** *baudRate_Bps*, **uint32_t** *srcClock_Hz*)

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>baudRate_Bps</i>	Desired baud rate.
<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.

16.6.7.13 void FLEXIO_MCULCD_SetSingleBeatWriteConfig ([FLEXIO_MCULCD_Type](#) * *base*)

At the begining 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 FLEXIO_MCULCD_Type .
-------------	---

Note

This is an internal used function, upper layer should not use.

16.6.7.14 void FLEXIO_MCULCD_ClearSingleBeatWriteConfig ([FLEXIO_MCULCD_Type](#) * *base*)

Clear the write configuration set by [FLEXIO_MCULCD_SetSingleBeatWriteConfig](#).

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
-------------	---

Note

This is an internal used function, upper layer should not use.

16.6.7.15 void FLEXIO_MCULCD_SetSingleBeatReadConfig (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_ClearSingleBeatReadConfig](#).

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
-------------	---

Note

This is an internal used function, upper layer should not use.

16.6.7.16 void FLEXIO_MCULCD_ClearSingleBeatReadConfig (FLEXIO_MCULCD_Type * *base*)

Clear the read configuration set by [FLEXIO_MCULCD_SetSingleBeatReadConfig](#).

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
-------------	---

Note

This is an internal used function, upper layer should not use.

16.6.7.17 void FLEXIO_MCULCD_SetMultiBeatsWriteConfig (FLEXIO_MCULCD_Type * *base*)

At the begining 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_ClearMultiBeatsWriteConfig](#).

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
-------------	---

Note

This is an internal used function, upper layer should not use.

16.6.7.18 void FLEXIO_MCULCD_ClearMultiBeatsWriteConfig (FLEXIO_MCULCD_Type * *base*)

Clear the write configuration set by FLEXIO_MCULCD_SetMultBeatsWriteConfig.

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
-------------	---

Note

This is an internal used function, upper layer should not use.

16.6.7.19 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_ClearMultBeatsReadConfig](#).

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
-------------	---

Note

This is an internal used function, upper layer should not use.

16.6.7.20 void FLEXIO_MCULCD_ClearMultiBeatsReadConfig (FLEXIO_MCULCD_Type * *base*)

Clear the read configuration set by [FLEXIO_MCULCD_SetMultBeatsReadConfig](#).

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
-------------	---

Note

This is an internal used function, upper layer should not use.

16.6.7.21 static void FLEXIO_MCULCD_Enable (FLEXIO_MCULCD_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
<i>enable</i>	True to enable, false does not have any effect.

16.6.7.22 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 FLEXIO_MCULCD_Type structure.
-------------	--

Returns

The data read out.

Note

Don't use this function with DMA APIs.

**16.6.7.23 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 FLEXIO_MCULCD_Type structure.
<i>data</i>	The data to write.

Note

Don't use this function with DMA APIs.

16.6.7.24 static void FLEXIO_MCULCD_StartTransfer (FLEXIO_MCULCD_Type * *base*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
-------------	--

16.6.7.25 static void FLEXIO_MCULCD_StopTransfer (FLEXIO_MCULCD_Type * *base*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
-------------	--

16.6.7.26 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.

16.6.7.27 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 FLEXIO_MCULCD_Type structure.
<i>command</i>	The command to send.

16.6.7.28 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 FLEXIO_MCULCD_Type structure.
<i>data</i>	The data array to send.
<i>size</i>	How many bytes to write.

16.6.7.29 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 FLEXIO_MCULCD_Type structure.
<i>data</i>	The array to save the data.
<i>size</i>	How many bytes to read.

16.6.7.30 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 FLEXIO_MCULCD_Type structure.
<i>sameValue</i>	The same value to send.
<i>size</i>	How many bytes to send.

16.6.7.31 void FLEXIO_MCULCD_TransferBlocking ([FLEXIO_MCULCD_Type](#) * *base*, [flexio_mculed_transfer_t](#) * *xfer*)

Note

The API does not return until the transfer finished.

Parameters

<i>base</i>	pointer to FLEXIO_MCULCD_Type structure.
<i>xfer</i>	pointer to flexio_mculed_transfer_t structure.

16.6.7.32 status_t FLEXIO_MCULCD_TransferCreateHandle ([FLEXIO_MCULCD_Type](#) * *base*, [flexio_mculed_handle_t](#) * *handle*, [flexio_mculed_transfer_callback_t](#) *callback*, [void](#) * *userData*)

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>handle</i>	Pointer to the flexio_mculed_handle_t 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.

16.6.7.33 status_t FLEXIO_MCULCD_TransferNonBlocking ([FLEXIO_MCULCD_Type](#) * *base*, [flexio_mculed_handle_t](#) * *handle*, [flexio_mculed_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 FLEXIO_MCULCD_Type structure.
<i>handle</i>	Pointer to the flexio_mculed_handle_t structure to store the transfer state.
<i>xfer</i>	FlexIO MCULCD transfer structure. See flexio_mculed_transfer_t .

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.

16.6.7.34 void FLEXIO_MCULCD_TransferAbort ([FLEXIO_MCULCD_Type](#) * *base*, [flexio_mculed_handle_t](#) * *handle*)

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>handle</i>	Pointer to the flexio_mculed_handle_t structure to store the transfer state.

16.6.7.35 status_t FLEXIO_MCULCD_TransferGetCount ([FLEXIO_MCULCD_Type](#) * *base*, [flexio_mculed_handle_t](#) * *handle*, [size_t](#) * *count*)

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>handle</i>	Pointer to the flexio_mculed_handle_t 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_NoTransferIn-Progress</i>	No transfer in process.

16.6.7.36 void FLEXIO_MCULCD_TransferHandleIRQ ([void](#) * *base*, [void](#) * *handle*)

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>handle</i>	Pointer to the <code>flexio_mculed_handle_t</code> structure to store the transfer state.

16.6.8 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.

16.6.8.1 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_mculedma_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_mculedma_transfer_callback_t](#))([FLEXIO_MCULCD_Type](#) *base, [flexio_mculedma_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_mculedma_handle_t](#) *handle, [flexio_mculedma_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_mculedma_handle_t](#) *handle, [flexio_mculedma_transfer_t](#) *xfer)
Performs a non-blocking FlexIO MCULCD transfer using eDMA.

- void **FLEXIO_MCULCD_TransferAbortEDMA** (**FLEXIO_MCULCD_Type** *base, **flexio_mculedma_handle_t** *handle)

Aborts a FlexIO MCULCD transfer using eDMA.
- **status_t FLEXIO_MCULCD_TransferGetCountEDMA** (**FLEXIO_MCULCD_Type** *base, **flexio_mculedma_handle_t** *handle, **size_t** *count)

Gets the remaining bytes for FlexIO MCULCD eDMA transfer.

16.6.8.2 Data Structure Documentation

16.6.8.2.1 struct _flexio_mculedma_handle

typedef for **flexio_mculedma_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_mculedma_transfer_callback_t completionCallback**

Callback for MCULCD DMA transfer.
- **void * userData**

User Data for MCULCD DMA callback.

Field Documentation

(1) **FLEXIO_MCULCD_Type* flexio_mculedma_handle_t::base**

- (2) `uint8_t flexio_mculedma_handle_t::txShifterNum`
- (3) `uint8_t flexio_mculedma_handle_t::rxShifterNum`
- (4) `uint32_t flexio_mculedma_handle_t::minorLoopBytes`
- (5) `edma_modulo_t flexio_mculedma_handle_t::txEdmaModulo`
- (6) `edma_modulo_t flexio_mculedma_handle_t::rxEdmaModulo`
- (7) `uint32_t flexio_mculedma_handle_t::dataAddrOrSameValue`

When writing or reading array, this is the address of the data array.

- (8) `size_t flexio_mculedma_handle_t::dataCount`
- (9) `volatile size_t flexio_mculedma_handle_t::remainingCount`
- (10) `volatile uint32_t flexio_mculedma_handle_t::state`

16.6.8.3 Macro Definition Documentation

16.6.8.3.1 #define FSL_FLEXIO_MCULCD_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

16.6.8.4 Typedef Documentation

16.6.8.4.1 typedef void(* flexio_mculedma_transfer_callback_t)(FLEXIO_MCULCD_Type **base*, flexio_mculedma_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.

16.6.8.5 Function Documentation

16.6.8.5.1 status_t FLEXIO_MCULCD_TransferCreateHandleEDMA (FLEXIO_MCULCD_Type * *base*, flexio_mculedma_handle_t * *handle*, flexio_mculedma_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 FLEXIO_MCULCD_Type structure.
<i>handle</i>	Pointer to flexio_mculedma_handle_t 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.
------------------------	---------------------------------

16.6.8.5.2 status_t FLEXIO_MCULCD_TransferEDMA (**FLEXIO_MCULCD_Type * *base*, **flexio_mculedma_handle_t** * *handle*, **flexio_mculed_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 FLEXIO_MCULCD_Type structure.
<i>handle</i>	pointer to flexio_mculedma_handle_t 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.

16.6.8.5.3 void FLEXIO_MCULCD_TransferAbortEDMA (**FLEXIO_MCULCD_Type * *base*, **flexio_mculedma_handle_t** * *handle*)**

Parameters

<i>base</i>	pointer to FLEXIO_MCULCD_Type structure.
<i>handle</i>	FlexIO MCULCD eDMA handle pointer.

16.6.8.5.4 status_t FLEXIO_MCULCD_TransferGetCountEDMA ([FLEXIO_MCULCD_Type](#) * *base*, [flexio_mculcd_edma_handle_t](#) * *handle*, [size_t](#) * *count*)

Parameters

<i>base</i>	pointer to FLEXIO_MCULCD_Type 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_NoTransferIn-Progress</i>	No transfer in process.

16.7 FlexIO SPI Driver

16.7.1 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.

16.7.2 Typical use case

16.7.2.1 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 = 5000000U;

    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)
    {
        // ...
    }
}

```

16.7.2.2 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** (0xFFFFU)
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.

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 }

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 = 0x1U,

kFLEXIO_SPI_8bitLsb = 0x2U,

kFLEXIO_SPI_16bitMsb = 0x9U,

kFLEXIO_SPI_16bitLsb = 0xaU }

Define FlexIO SPI transfer flags.

Driver version

- #define **FSL_FLEXIO_SPI_DRIVER_VERSION** (MAKE_VERSION(2, 2, 1))

FlexIO SPI driver version 2.2.1.

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, uint16_t data)
Writes one byte of data, which is sent using the MSB method.
- static uint16_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.

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.

16.7.3 Data Structure Documentation

16.7.3.1 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`
- (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]`

16.7.3.2 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`

16.7.3.3 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`

16.7.3.4 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`

16.7.3.5 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`

16.7.4 Macro Definition Documentation

16.7.4.1 `#define FSL_FLEXIO_SPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))`

16.7.4.2 `#define FLEXIO_SPI_DUMMYDATA (0xFFFFU)`

16.7.4.3 `#define SPI_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`

16.7.5 Typedef Documentation

16.7.5.1 `typedef flexio_spi_master_handle_t flexio_spi_slave_handle_t`

16.7.6 Enumeration Type Documentation

16.7.6.1 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.

16.7.6.2 enum flexio_spi_clock_phase_t

Enumerator

kFLEXIO_SPI_ClockPhaseFirstEdge First edge on SPSCK occurs at the middle of the first cycle of a data transfer.

kFLEXIO_SPI_ClockPhaseSecondEdge First edge on SPSCK occurs at the start of the first cycle of a data transfer.

16.7.6.3 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.

16.7.6.4 enum flexio_spi_data_bitcount_mode_t

Enumerator

kFLEXIO_SPI_8BitMode 8-bit data transmission mode.

kFLEXIO_SPI_16BitMode 16-bit data transmission mode.

16.7.6.5 enum _flexio_spi_interrupt_enable

Enumerator

kFLEXIO_SPI_TxEmptyInterruptEnable Transmit buffer empty interrupt enable.

kFLEXIO_SPI_RxFullInterruptEnable Receive buffer full interrupt enable.

16.7.6.6 enum _flexio_spi_status_flags

Enumerator

kFLEXIO_SPI_TxBufferEmptyFlag Transmit buffer empty flag.

kFLEXIO_SPI_RxBufferFullFlag Receive buffer full flag.

16.7.6.7 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.

16.7.6.8 enum _flexio_spi_transfer_flags

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.

16.7.7 Function Documentation

16.7.7.1 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 $2 \times 2 = 4$. If FlexIO SPI master communicates with FlexIO SPI slave, the maximum baud rate is FlexIO clock frequency divided by $(1.5 + 2.5) \times 2 = 8$.

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 FLEXIO_SPI_Type structure.
<i>masterConfig</i>	Pointer to the flexio_spi_master_config_t structure.
<i>srcClock_Hz</i>	FlexIO source clock in Hz.

16.7.7.2 void [FLEXIO_SPI_MasterDeinit](#) ([FLEXIO_SPI_Type](#) * *base*)

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type .
-------------	--

16.7.7.3 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 flexio_spi_master_config_t structure.
---------------------	--

16.7.7.4 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 FLEXIO_SPI_Type structure.
<i>slaveConfig</i>	Pointer to the flexio_spi_slave_config_t structure.

16.7.7.5 void FLEXIO_SPI_SlaveDeinit (**FLEXIO_SPI_Type** * *base*)

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type .
-------------	--

16.7.7.6 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 flexio_spi_slave_config_t structure.
--------------------	---

16.7.7.7 uint32_t FLEXIO_SPI_GetStatusFlags ([FLEXIO_SPI_Type](#) * *base*)

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
-------------	---

Returns

status flag; Use the status flag to AND the following flag mask and get the status.

- [kFLEXIO_SPI_TxEmptyFlag](#)
- [kFLEXIO_SPI_RxEmptyFlag](#)

16.7.7.8 void FLEXIO_SPI_ClearStatusFlags ([FLEXIO_SPI_Type](#) * *base*, [uint32_t](#) *mask*)

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>mask</i>	status flag The parameter can be any combination of the following values: <ul style="list-style-type: none">• kFLEXIO_SPI_TxEmptyFlag• kFLEXIO_SPI_RxEmptyFlag

16.7.7.9 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 FLEXIO_SPI_Type structure.
<i>mask</i>	interrupt source. The parameter can be any combination of the following values: <ul style="list-style-type: none"> • kFLEXIO_SPI_RxFullInterruptEnable • kFLEXIO_SPI_TxEmptyInterruptEnable

16.7.7.10 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 FLEXIO_SPI_Type structure.
<i>mask</i>	interrupt source The parameter can be any combination of the following values: <ul style="list-style-type: none"> • kFLEXIO_SPI_RxFullInterruptEnable • kFLEXIO_SPI_TxEmptyInterruptEnable

16.7.7.11 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 FLEXIO_SPI_Type structure.
<i>mask</i>	SPI DMA source.
<i>enable</i>	True means enable DMA, false means disable DMA.

16.7.7.12 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 FLEXIO_SPI_Type structure.
<i>direction</i>	Shift direction of MSB first or LSB first.

Returns

FlexIO SPI transmit data register address.

16.7.7.13 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 FLEXIO_SPI_Type structure.
<i>direction</i>	Shift direction of MSB first or LSB first.

Returns

FlexIO SPI receive data register address.

16.7.7.14 static void FLEXIO_SPI_Enable ([FLEXIO_SPI_Type](#) * *base*, [bool](#) *enable*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type .
<i>enable</i>	True to enable, false does not have any effect.

16.7.7.15 void FLEXIO_SPI_MasterSetBaudRate ([FLEXIO_SPI_Type](#) * *base*, [uint32_t](#) *baudRate_Bps*, [uint32_t](#) *srcClockHz*)

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>baudRate_Bps</i>	Baud Rate needed in Hz.
<i>srcClockHz</i>	SPI source clock frequency in Hz.

16.7.7.16 static void FLEXIO_SPI_WriteData (FLEXIO_SPI_Type * *base*, flexio_spi_shift_direction_t *direction*, uint16_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 FLEXIO_SPI_Type structure.
<i>direction</i>	Shift direction of MSB first or LSB first.
<i>data</i>	8 bit/16 bit data.

16.7.7.17 static uint16_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 FLEXIO_SPI_Type structure.
<i>direction</i>	Shift direction of MSB first or LSB first.

Returns

8 bit/16 bit data received.

16.7.7.18 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 FLEXIO_SPI_Type 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.

16.7.7.19 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 FLEXIO_SPI_Type 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.

16.7.7.20 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 FLEXIO_SPI_Type structure
<i>xfer</i>	FlexIO SPI transfer structure, see flexio_spi_transfer_t .

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_FLEXIO_SPI_Timeout</i>	The transfer timed out and was aborted.

16.7.7.21 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 FLEXIO_SPI_Type structure.
<i>handle</i>	Pointer to the flexio_spi_master_handle_t 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.

16.7.7.22 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 FLEXIO_SPI_Type 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 flexio_spi_transfer_t .

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.

**16.7.7.23 void FLEXIO_SPI_MasterTransferAbort (`FLEXIO_SPI_Type * base,`
`flexio_spi_master_handle_t * handle`)**

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>handle</i>	Pointer to the <code>flexio_spi_master_handle_t</code> structure to store the transfer state.

**16.7.7.24 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 FLEXIO_SPI_Type 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.

**16.7.7.25 void FLEXIO_SPI_MasterTransferHandleIRQ (`void * spiType, void * spiHandle`
`)`**

Parameters

<i>spiType</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>spiHandle</i>	Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

16.7.7.26 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 FLEXIO_SPI_Type structure.
<i>handle</i>	Pointer to the flexio_spi_slave_handle_t 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.

16.7.7.27 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 flexio_spi_slave_handle_t structure to store the transfer state.
<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>xfer</i>	FlexIO SPI transfer structure. See flexio_spi_transfer_t .

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.

16.7.7.28 static void FLEXIO_SPI_SlaveTransferAbort (FLEXIO_SPI_Type * *base*, flexio_spi_slave_handle_t * *handle*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>handle</i>	Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

16.7.7.29 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 FLEXIO_SPI_Type structure.
<i>handle</i>	Pointer to the flexio_spi_slave_handle_t 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.

16.7.7.30 void FLEXIO_SPI_SlaveTransferHandleIRQ (void * *spiType*, void * *spiHandle*)

Parameters

<i>spiType</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>spiHandle</i>	Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

16.7.8 FlexIO eDMA SPI Driver

16.7.8.1 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, 2, 1))`
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.

16.7.8.2 Data Structure Documentation

16.7.8.2.1 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**

16.7.8.3 Macro Definition Documentation

16.7.8.3.1 #define FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))

16.7.8.4 Typedef Documentation

16.7.8.4.1 `typedef flexio_spi_master_edma_handle_t flexio_spi_slave_edma_handle_t`

16.7.8.5 Function Documentation

16.7.8.5.1 `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 FLEXIO_SPI_Type 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.

16.7.8.5.2 `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 FLEXIO_SPI_Type structure.
-------------	---

<i>handle</i>	Pointer to flexio_spi_master_edma_handle_t 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.

16.7.8.5.3 void FLEXIO_SPI_MasterTransferAbortEDMA (FLEXIO_SPI_Type * *base*, flexio_spi_master_edma_handle_t * *handle*)

Parameters

<i>base</i>	Pointer to FLEXIO_SPI_Type structure.
<i>handle</i>	FlexIO SPI eDMA handle pointer.

16.7.8.5.4 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 FLEXIO_SPI_Type structure.
<i>handle</i>	FlexIO SPI eDMA handle pointer.
<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

16.7.8.5.5 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 FLEXIO_SPI_Type 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.

16.7.8.5.6 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 FLEXIO_SPI_Type 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.

16.7.8.5.7 static void FLEXIO_SPI_SlaveTransferAbortEDMA ([FLEXIO_SPI_Type](#) * *base*, `flexio_spi_slave_edma_handle_t` * *handle*) [inline], [static]

Parameters

<i>base</i>	Pointer to FLEXIO_SPI_Type structure.
<i>handle</i>	Pointer to <code>flexio_spi_slave_edma_handle_t</code> structure to store the transfer state.

16.7.8.5.8 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 FLEXIO_SPI_Type structure.
<i>handle</i>	FlexIO SPI eDMA handle pointer.
<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

16.8 FlexIO UART Driver

16.8.1 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 `kStatus_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.

16.8.2 Typical use case

16.8.2.1 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);
}

```

16.8.2.2 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, 1200000000U);
//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)
{
}

// ...
}

```

16.8.2.3 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)
        {
        }
    }
}

// ...
}

```

16.8.2.4 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);

```

```

    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.

16.8.3 Data Structure Documentation

16.8.3.1 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]`

16.8.3.2 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**

16.8.3.3 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**

16.8.3.4 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`

16.8.4 Macro Definition Documentation

16.8.4.1 #define FSL_FLEXIO_UART_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))

16.8.4.2 #define UART_RETRY_TIMES 0U /* Defining to zero means to keep waiting for the flag until it is assert/deassert. */

16.8.5 Typedef Documentation

16.8.5.1 `typedef void(* flexio_uart_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, status_t status, void *userData)`

16.8.6 Enumeration Type Documentation

16.8.6.1 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.

16.8.6.2 `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

16.8.6.3 `enum _flexio_uart_interrupt_enable`

Enumerator

- kFLEXIO_UART_TxDataRegEmptyInterruptEnable* Transmit buffer empty interrupt enable.
- kFLEXIO_UART_RxDataRegFullInterruptEnable* Receive buffer full interrupt enable.

16.8.6.4 `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.

16.8.7 Function Documentation

16.8.7.1 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 FLEXIO_UART_Type structure.
<i>userConfig</i>	Pointer to the flexio_uart_config_t structure.
<i>srcClock_Hz</i>	FlexIO source clock in Hz.

Return values

<i>kStatus_Success</i>	Configuration success.
<i>kStatus_FLEXIO_UART-BaudrateNotSupport</i>	Baudrate is not supported for current clock source frequency.

16.8.7.2 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 FLEXIO_UART_Type structure
-------------	---

16.8.7.3 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 flexio_uart_config_t structure.
-------------------	--

16.8.7.4 uint32_t FLEXIO_UART_GetStatusFlags ([FLEXIO_UART_Type](#) * *base*)

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
-------------	--

Returns

FlexIO UART status flags.

16.8.7.5 void FLEXIO_UART_ClearStatusFlags ([FLEXIO_UART_Type](#) * *base*, uint32_t *mask*)

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
<i>mask</i>	Status flag. The parameter can be any combination of the following values: <ul style="list-style-type: none">• kFLEXIO_UART_TxDataRegEmptyFlag• kFLEXIO_UART_RxEmptyFlag• kFLEXIO_UART_RxOverRunFlag

16.8.7.6 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 FLEXIO_UART_Type structure.
<i>mask</i>	Interrupt source.

16.8.7.7 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 FLEXIO_UART_Type structure.
<i>mask</i>	Interrupt source.

16.8.7.8 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 FLEXIO_UART_Type structure.
-------------	--

Returns

FlexIO UART transmit data register address.

16.8.7.9 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 FLEXIO_UART_Type structure.
-------------	--

Returns

FlexIO UART receive data register address.

16.8.7.10 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 FLEXIO_UART_Type structure.
<i>enable</i>	True to enable, false to disable.

16.8.7.11 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 FLEXIO_UART_Type structure.
<i>enable</i>	True to enable, false to disable.

16.8.7.12 static void FLEXIO_UART_Enable ([FLEXIO_UART_Type](#) * *base*, *bool enable*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type .
<i>enable</i>	True to enable, false does not have any effect.

16.8.7.13 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 FLEXIO_UART_Type structure.
-------------	--

<i>buffer</i>	The data bytes to send.
---------------	-------------------------

16.8.7.14 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 FLEXIO_UART_Type structure.
<i>buffer</i>	The buffer to store the received bytes.

16.8.7.15 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 FLEXIO_UART_Type 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_Timeout</i>	Transmission timed out and was aborted.
<i>kStatus_Success</i>	Successfully wrote all data.

16.8.7.16 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 FLEXIO_UART_Type 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_Timeout</i>	Transmission timed out and was aborted.
<i>kStatus_Success</i>	Successfully received all data.

16.8.7.17 `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 FLEXIO_UART_Type 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.

16.8.7.18 `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 <code>FLEXIO_UART_Type</code> 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.

16.8.7.19 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 <code>FLEXIO_UART_Type</code> structure.
<i>handle</i>	Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state.

16.8.7.20 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 FLEXIO_UART_Type 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 flexio_uart_transfer_t .

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.

16.8.7.21 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 FLEXIO_UART_Type structure.
<i>handle</i>	Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state.

16.8.7.22 `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 FLEXIO_UART_Type 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.

16.8.7.23 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 FLEXIO_UART_Type structure.
<i>handle</i>	Pointer to the flexio_uart_handle_t structure to store the transfer state.
<i>xfer</i>	UART transfer structure. See flexio_uart_transfer_t .
<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.

16.8.7.24 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 FLEXIO_UART_Type structure.
<i>handle</i>	Pointer to the flexio_uart_handle_t structure to store the transfer state.

**16.8.7.25 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 FLEXIO_UART_Type structure.
<i>handle</i>	Pointer to the flexio_uart_handle_t structure to store the transfer state.
<i>count</i>	Number of bytes received 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.

16.8.7.26 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 FLEXIO_UART_Type structure.
<i>uartHandle</i>	Pointer to the flexio_uart_handle_t structure to store the transfer state.

16.8.8 FlexIO eDMA UART Driver

16.8.8.1 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.

16.8.8.2 Data Structure Documentation

16.8.8.2.1 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**

16.8.8.3 Macro Definition Documentation

16.8.8.3.1 #define FSL_FLEXIO_UART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 1))

16.8.8.4 Typedef Documentation

16.8.8.4.1 `typedef void(* flexio_uart_edma_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_edma_handle_t *handle, status_t status, void *userData)`

16.8.8.5 Function Documentation

16.8.8.5.1 `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 FLEXIO_UART_Type .
<i>handle</i>	Pointer to <code>flexio_uart_edma_handle_t</code> 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.

16.8.8.5.2 `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 FLEXIO_UART_Type
<i>handle</i>	UART handle pointer.
<i>xfer</i>	UART eDMA transfer structure, see flexio_uart_transfer_t .

Return values

<i>kStatus_Success</i>	if succeed, others failed.
<i>kStatus_FLEXIO_UART_TxBusy</i>	Previous transfer on going.

16.8.8.5.3 status_t FLEXIO_UART_TransferReceiveEDMA (***base***, ***handle***, ***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 FLEXIO_UART_Type
<i>handle</i>	Pointer to flexio_uart_edma_handle_t structure
<i>xfer</i>	UART eDMA transfer structure, see flexio_uart_transfer_t .

Return values

<i>kStatus_Success</i>	if succeed, others failed.
<i>kStatus_UART_RxBusy</i>	Previous transfer on going.

16.8.8.5.4 void FLEXIO_UART_TransferAbortSendEDMA (***base***, ***handle***)

This function aborts sent data which using eDMA.

Parameters

<i>base</i>	Pointer to FLEXIO_UART_Type
<i>handle</i>	Pointer to flexio_uart_edma_handle_t structure

16.8.8.5.5 void FLEXIO_UART_TransferAbortReceiveEDMA (***base***, ***handle***)

This function aborts the receive data which using eDMA.

Parameters

<i>base</i>	Pointer to FLEXIO_UART_Type
<i>handle</i>	Pointer to <code>flexio_uart_edma_handle_t</code> structure

16.8.8.5.6 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 FLEXIO_UART_Type
<i>handle</i>	Pointer to <code>flexio_uart_edma_handle_t</code> structure
<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.

16.8.8.5.7 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 FLEXIO_UART_Type
<i>handle</i>	Pointer to <code>flexio_uart_edma_handle_t</code> structure
<i>count</i>	Number of bytes received 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.
------------------------	--------------------------------

Chapter 17

FLEXRAM: on-chip RAM manager

17.1 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`.

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_Read = 0U, kFLEXRAM_Write = 1U }`
Flexram write/read selection.
- `enum { kFLEXRAM_OCRAMAccessError = FLEXRAM_INT_STATUS_OCRAM_ERR_STATUS_M-ASK, kFLEXRAM_DTCMAccessError = FLEXRAM_INT_STATUS_DTCM_ERR_STATUS_MASK, kFLEXRAM_ITCMAccessError = FLEXRAM_INT_STATUS_ITCM_ERR_STATUS_MASK, kFLEXRAM_OCRAMMagicAddrMatch = FLEXRAM_INT_STATUS_OCRAM_MAM_STAT-US_MASK, kFLEXRAM_DTCMMagicAddrMatch = FLEXRAM_INT_STATUS_DTCM_MAM_STATUS_-MASK, kFLEXRAM_ITCMMagicAddrMatch = FLEXRAM_INT_STATUS_ITCM_MAM_STATUS_M-ASK }`
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

- 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.
- static void **FLEXRAM_SetOCRAMMagicAddr** (FLEXRAM_Type *base, uint16_t magicAddr, uint32_t rwSel)
FLEXRAM OCRAM magic addr configuration.
- static void **FLEXRAM_SetDTCMMagicAddr** (FLEXRAM_Type *base, uint16_t magicAddr, uint32_t rwSel)
FLEXRAM DTCM magic addr configuration.
- static void **FLEXRAM_SetITCMMagicAddr** (FLEXRAM_Type *base, uint16_t magicAddr, uint32_t rwSel)
FLEXRAM ITCM magic addr configuration.

Driver version

- #define **FSL_FLEXRAM_DRIVER_VERSION** (MAKE_VERSION(2U, 1U, 0U))
Driver version 2.1.0.

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.

17.2 Macro Definition Documentation

17.2.1 #define FSL_FLEXRAM_DRIVER_VERSION (MAKE_VERSION(2U, 1U, 0U))

17.2.2 #define FLEXRAM_ECC_ERROR_DETAILED_INFO 0U /* Define to zero means get raw ECC error information, which needs parse it by user. */

17.3 Enumeration Type Documentation

17.3.1 anonymous enum

Enumerator

kFLEXRAM_Read read
kFLEXRAM_Write write

17.3.2 anonymous enum

Enumerator

kFLEXRAM_OCRAMAccessError OCRAM accesses unallocated address.
kFLEXRAM_DTCMAccessError DTCM accesses unallocated address.
kFLEXRAM_ITCMAccessError ITCM accesses unallocated address.
kFLEXRAM_OCRAMMagicAddrMatch OCRAM magic address match.
kFLEXRAM_DTCMMagicAddrMatch DTCM magic address match.
kFLEXRAM_ITCMMagicAddrMatch ITCM magic address match.

17.3.3 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 better.

Enumerator

kFLEXRAM_TCMAccessFastMode fast access mode
kFLEXRAM_TCMAccessWaitMode wait access mode

17.3.4 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.

17.4 Function Documentation

17.4.1 void FLEXRAM_Init (**FLEXRAM_Type * base**)

Parameters

<i>base</i>	FLEXRAM base address.
-------------	-----------------------

17.4.2 static uint32_t FLEXRAM_GetInterruptStatus (**FLEXRAM_Type * base**) [inline], [static]

Parameters

<i>base</i>	FLEXRAM base address.
-------------	-----------------------

17.4.3 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.

17.4.4 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.

17.4.5 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.

17.4.6 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.

17.4.7 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.

17.4.8 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.

17.4.9 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.

17.4.10 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.

17.4.11 static void FLEXRAM_SetOCRAMMMagicAddr (FLEXRAM_Type * *base*, uint16_t *magicAddr*, uint32_t *rwSel*) [inline], [static]

When read/write access hit magic address, it will generate interrupt.

Parameters

<i>base</i>	FLEXRAM base address.
<i>magicAddr</i>	Magic address, the actual address bits [18:3] is corresponding to the register field [16:1].

<i>rwSel</i>	Read/write selection. 0 for read access while 1 for write access.
--------------	---

17.4.12 static void FLEXRAM_SetDTCMMagicAddr (FLEXRAM_Type * *base*, uint16_t *magicAddr*, uint32_t *rwSel*) [inline], [static]

When read/write access hits magic address, it will generate interrupt.

Parameters

<i>base</i>	FLEXRAM base address.
<i>magicAddr</i>	Magic address, the actual address bits [18:3] is corresponding to the register field [16:1].
<i>rwSel</i>	Read/write selection. 0 for read access while 1 write access.

17.4.13 static void FLEXRAM_SetITCMMagicAddr (FLEXRAM_Type * *base*, uint16_t *magicAddr*, uint32_t *rwSel*) [inline], [static]

When read/write access hits magic address, it will generate interrupt.

Parameters

<i>base</i>	FLEXRAM base address.
<i>magicAddr</i>	Magic address, the actual address bits [18:3] is corresponding to the register field [16:1].
<i>rwSel</i>	Read/write selection. 0 for read access while 1 for write access.

Chapter 18

FLEXSPI: Flexible Serial Peripheral Interface Driver

18.1 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.

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_AhbBusErrorFlag = FLEXSPI_INTEN_AHBBUSERROREN_MASK,

kFLEXSPI_SckStoppedBecauseTxEmptyFlag,

kFLEXSPI_SckStoppedBecauseRxFullFlag,

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 = 0xFFFFU }

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 }
```
- enum `flexspi_arb_command_source_t`

FLEXSPI operation port select.
- enum `flexspi_command_type_t`

Trigger source of current command sequence granted by arbitrator.

```
kFLEXSPI_Command,
kFLEXSPI_Config }
```

Command type.

Driver version

- #define `FSL_FLEXSPI_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 5)`)
FLEXSPI driver version 2.3.5.

Initialization and deinitialization

- `uint32_t FLEXSPIGetInstance (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 **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, uint32_t *buffer, size_t size)
Sends a buffer of data bytes using blocking method.
- status_t [FLEXSPI_ReadBlocking](#) (FLEXSPI_Type *base, uint32_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.

18.2 Data Structure Documentation

18.2.1 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**

18.2.2 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`

18.2.3 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`

18.2.4 struct _flexspi_handle

Data Fields

- `uint32_t state`
Internal state for FLEXSPI transfer.
- `uint32_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) `uint32_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`

18.3 Macro Definition Documentation

18.3.1 #define FSL_FLEXSPI_DRIVER_VERSION (MAKE_VERSION(2, 3, 5))

18.3.2 #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))
```

18.4 Typedef Documentation

18.4.1 **typedef void(* flexspi_transfer_callback_t)(FLEXSPI_Type *base, flexspi_handle_t *handle, status_t status, void *userData)**

18.5 Enumeration Type Documentation

18.5.1 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.

18.5.2 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.

18.5.3 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].

18.5.4 enum flexspi_flags_t

Enumerator

- kFLEXSPI_SequenceExecutionTimeoutFlag*** Sequence execution timeout.
- kFLEXSPI_AhbBusErrorFlag*** AHB Bus error flag.
- 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_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.

18.5.5 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.

18.5.6 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.

18.5.7 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.

18.5.8 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_IpCmdErrorFlashBoundaryAcrossss Flash boundary crossed.

18.5.9 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.

18.5.10 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.

18.5.11 enum flexspi_arb_command_source_t

18.5.12 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.

18.6 Function Documentation

18.6.1 uint32_t FLEXSPI_GetInstance (**FLEXSPI_Type** * *base*)

Parameters

<i>base</i>	FLEXSPI base pointer.
-------------	-----------------------

18.6.2 status_t FLEXSPI_CheckAndClearError (**FLEXSPI_Type** * *base*, **uint32_t** *status*)

Parameters

<i>base</i>	FLEXSPI base pointer.
<i>status</i>	interrupt status.

18.6.3 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.

18.6.4 void FLEXSPI_GetDefaultConfig (**flexspi_config_t** * *config*)

Parameters

<i>config</i>	FLEXSPI configuration structure.
---------------	----------------------------------

18.6.5 void FLEXSPI_Deinit (**FLEXSPI_Type** * *base*)

Clears the FLEXSPI state and FLEXSPI module registers.

Parameters

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

18.6.6 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.

18.6.7 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.

18.6.8 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.
-------------	----------------------------------

**18.6.9 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.

**18.6.10 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.

**18.6.11 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.

**18.6.12 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.

**18.6.13 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.

**18.6.14 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.
------------	------------------

**18.6.15 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.
------------	------------------

18.6.16 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.

18.6.17 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.

18.6.18 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 flexspi_flags_t could get the related status.
------------------	---

18.6.19 **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.

18.6.20 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.
----------------	-------------------------------------

18.6.21 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.
--------------	--------------------------------------

18.6.22 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.
--------------	---------------------------------------

**18.6.23 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.

**18.6.24 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 flexspi_read_sample_clock_t

**18.6.25 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.

**18.6.26 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.

18.6.27 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.

18.6.28 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.

18.6.29 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.

18.6.30 status_t FLEXSPI_WriteBlocking (*FLEXSPI_Type * base*, *uint32_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

18.6.31 status_t FLEXSPI_ReadBlocking (*FLEXSPI_Type * base*, *uint32_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_IpCommandSequenceError</i>	IP command sequence error detected
<i>kStatus_FLEXSPI_IpCommandGrantTimeout</i>	IP command grant timeout detected

18.6.32 **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_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

18.6.33 **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 <code>flexspi_handle_t</code> structure to store the transfer state.
<i>callback</i>	pointer to user callback function.
<i>userData</i>	user parameter passed to the callback function.

18.6.34 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

<code>kStatus_Success</code>	Successfully start the data transmission.
<code>kStatus_FLEXSPI_Busy</code>	Previous transmission still not finished.

18.6.35 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.

18.6.36 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

18.6.37 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.

Chapter 19

GPC: General Power Controller Driver

19.1 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_RequestPdram0PowerDown](#) (GPC_Type *base, bool enable)
FLEXRAM PDRAM0 Power Gate Enable.
- 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.

19.2 Macro Definition Documentation

19.2.1 #define [FSL_GPC_DRIVER_VERSION](#) (MAKE_VERSION(2, 1, 1))

19.3 Function Documentation

19.3.1 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.

19.3.2 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.

19.3.3 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.

19.3.4 static void GPC_RequestPdram0PowerDown (GPC_Type * *base*, bool *enable*) [inline], [static]

This function configures the FLEXRAM PDRAM0 if it will keep power when cpu core is power down. When the PDRAM0 Power is 1, PDRAM0 will be power down once when CPU core is power down. When the PDRAM0 Power is 0, PDRAM0 will keep power on even if CPU core is power down. When CPU core is re-power up, the default setting is 1.

Parameters

<i>base</i>	GPC peripheral base address.
<i>enable</i>	Enable the request or not.

19.3.5 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 20

GPT: General Purpose Timer

20.1 Overview

The MCUXpresso SDK provides a driver for the General Purpose Timer (GPT) of MCUXpresso SDK devices.

20.2 Function groups

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

20.2.1 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.

20.3 Typical use case

20.3.1 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, 3)`)

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)

- static `gpt_output_operation_mode_t GPT_GetOutputOperationMode` (`GPT_Type *base, gpt_output_compare_channel_t channel`)

Set GPT operation mode of output compare channel.
- static `void GPT_SetOutputCompareValue` (`GPT_Type *base, gpt_output_compare_channel_t channel, uint32_t value`)

Get GPT operation mode of output compare channel.
- static `uint32_t GPT_GetOutputCompareValue` (`GPT_Type *base, gpt_output_compare_channel_t channel`)

Set GPT output compare value of output compare channel.
- static `void GPT_ForceOutput` (`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.

20.4 Data Structure Documentation

20.4.1 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`

20.5 Enumeration Type Documentation**20.5.1 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.

20.5.2 enum gpt_input_capture_channel_t

Enumerator

- kGPT_InputCapture_Channel1* GPT Input Capture Channel1.
- kGPT_InputCapture_Channel2* GPT Input Capture Channel2.

20.5.3 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.

20.5.4 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.

20.5.5 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.

20.5.6 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.

20.5.7 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.

20.6 Function Documentation

20.6.1 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.

20.6.2 void GPT_Deinit (*GPT_Type* * *base*)

Parameters

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

20.6.3 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.
---------------	--

20.6.4 static void GPT_SoftwareReset (**GPT_Type** * *base*) [inline], [static]

Parameters

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

20.6.5 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 gpt_clock_source_t typedef enumeration).

20.6.6 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).

20.6.7 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).

20.6.8 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).

20.6.9 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).

20.6.10 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).

20.6.11 static void GPT_StartTimer (**GPT_Type** * *base*) [inline], [static]

Parameters

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

20.6.12 static void GPT_StopTimer (**GPT_Type** * *base*) [inline], [static]

Parameters

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

20.6.13 static uint32_t GPT_GetCurrentTimerCount (**GPT_Type** * *base*) [inline], [static]

Parameters

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

Returns

Current GPT counter value.

20.6.14 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 gpt_input_capture_channel_t typedef enumeration).
<i>mode</i>	GPT input capture operation mode (see gpt_input_operation_mode_t typedef enumeration).

20.6.15 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 gpt_input_capture_channel_t typedef enumeration).

Returns

GPT input capture operation mode (see [gpt_input_operation_mode_t](#) typedef enumeration).

20.6.16 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 gpt_input_capture_channel_t typedef enumeration).

Returns

GPT input capture value.

20.6.17 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 gpt_output_compare_channel_t typedef enumeration).
<i>mode</i>	GPT output operation mode (see gpt_output_operation_mode_t typedef enumeration).

20.6.18 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 gpt_output_compare_channel_t typedef enumeration).

Returns

GPT output operation mode (see [gpt_output_operation_mode_t](#) typedef enumeration).

20.6.19 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 gpt_output_compare_channel_t typedef enumeration).
<i>value</i>	GPT output compare value.

20.6.20 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 gpt_output_compare_channel_t typedef enumeration).

Returns

GPT output compare value.

20.6.21 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 gpt_output_compare_channel_t typedef enumeration).

20.6.22 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 gpt_interrupt_enable_t

20.6.23 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 gpt_interrupt_enable_t

20.6.24 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](#)

20.6.25 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 gpt_status_flag_t for bit definition).

Returns

GPT status, each bit represents one status flag.

20.6.26 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 gpt_status_flag_t for bit definition).

Chapter 21

GPIO: General-Purpose Input/Output Driver

21.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General-Purpose Input/Output (GPIO) module of MCUXpresso SDK devices.

21.2 Typical use case

21.2.1 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, 5)`)
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.

21.3 Data Structure Documentation

21.3.1 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`

21.4 Macro Definition Documentation

21.4.1 `#define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))`

21.5 Enumeration Type Documentation

21.5.1 enum gpio_pin_direction_t

Enumerator

kGPIO_DigitalInput Set current pin as digital input.

kGPIO_DigitalOutput Set current pin as digital output.

21.5.2 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.

21.6 Function Documentation

21.6.1 `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 gpio_pin_config_t structure that contains the configuration information.

21.6.2 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>	GPIOin output logic level. <ul style="list-style-type: none"> • 0: corresponding pin output low-logic level. • 1: corresponding pin output high-logic level.

21.6.3 static void GPIO_WritePinOutput (**GPIO_Type** * *base*, **uint32_t** *pin*, **uint8_t** *output*) [inline], [static]

Deprecated Do not use this function. It has been superceded by [GPIO_PinWrite](#).

21.6.4 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

21.6.5 static void GPIO_SetPinsOutput (**GPIO_Type** * *base*, **uint32_t** *mask*) [inline], [static]

Deprecated Do not use this function. It has been superceded by [GPIO_PortSet](#).

21.6.6 **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

21.6.7 static void GPIO_ClearPinsOutput (**GPIO_Type * *base*, **uint32_t** *mask*)
[inline], [static]**

Deprecated Do not use this function. It has been superceded by [GPIO_PortClear](#).

21.6.8 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

21.6.9 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.
-------------	-------------------

21.6.10 static uint32_t GPIO_ReadPinInput (**GPIO_Type * *base*, **uint32_t** *pin*)
[inline], [static]**

Deprecated Do not use this function. It has been superceded by [GPIO_PinRead](#).

21.6.11 **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.
-------------	-----------------------

21.6.12 static uint8_t GPIO_ReadPadStatus (*GPIO_Type* * *base*, *uint32_t* *pin*) [inline], [static]

Deprecated Do not use this function. It has been superceded by [GPIO_PinReadPadStatus](#).

21.6.13 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-Mode</i>	pointer to a gpio_interrupt_mode_t structure that contains the interrupt mode information.

21.6.14 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 superceded by [GPIO_PinSetInterruptConfig](#).

21.6.15 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.

**21.6.16 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.

**21.6.17 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.

**21.6.18 static void GPIO_DisableInterrupts (GPIO_Type * *base*, uint32_t *mask*)
[inline], [static]**

Deprecated Do not use this function. It has been superceded by [GPIO_PortDisableInterrupts](#).

**21.6.19 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.
----------------	----------------------------

21.6.20 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.
----------------	----------------------------

21.6.21 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.

21.6.22 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 22

KPP: KeyPad Port Driver

22.1 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.

22.2 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.

22.3 Data Structure Documentation

22.3.1 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".

22.4 Macro Definition Documentation

22.4.1 #define FSL_KPP_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

22.5 Enumeration Type Documentation

22.5.1 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.

22.5.2 enum kpp_sync_operation_t

Enumerator

- kKPP_ClearKeyDepressSyncChain* Keypad depress interrupt status.
- kKPP_SetKeyReleasesSyncChain* Keypad release interrupt status.

22.6 Function Documentation

22.6.1 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.

22.6.2 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.
-------------	------------------------------

22.6.3 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.

22.6.4 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.

22.6.5 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.

22.6.6 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.
-------------	--

22.6.7 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.

22.6.8 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 23

LPI2C: Low Power Inter-Integrated Circuit Driver

23.1 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, 3, 1))`
LPI2C driver version.

23.2 Macro Definition Documentation

23.2.1 `#define FSL_LPI2C_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))`

23.2.2 #define I2C_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */

23.3 Enumeration Type Documentation

23.3.1 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.

23.4 LPI2C Master Driver

23.4.1 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.

23.4.2 Data Structure Documentation

23.4.2.1 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`

23.4.2.2 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`

23.4.2.3 struct _lpi2c_master_transfer

This structure is used to pass transaction parameters to the [LPI2C_MasterTransferNonBlocking\(\)](#) API.

Data Fields

- `uint32_t flags`

- **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**

23.4.2.4 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`

23.4.3 Typedef Documentation

23.4.3.1 `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

<code>base</code>	The LPI2C peripheral base address.
<code>completion-Status</code>	Either kStatus_Success or an error code describing how the transfer completed.

<i>userData</i>	Arbitrary pointer-sized value passed from the application.
-----------------	--

23.4.4 Enumeration Type Documentation

23.4.4.1 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.

23.4.4.2 enum lpi2c_direction_t

Enumerator

- kLPI2C_Write* Master transmit.
- kLPI2C_Read* Master receive.

23.4.4.3 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)

23.4.4.4 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.

23.4.4.5 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.

23.4.4.6 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.

23.4.4.7 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.

23.4.5 Function Documentation

23.4.5.1 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 lpi2c_master_config_t .
-----	---------------------	--

23.4.5.2 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 LPI2C_MasterGetDefaultConfig() 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.

23.4.5.3 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.
-------------	------------------------------------

23.4.5.4 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.

23.4.5.5 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.
-------------	------------------------------------

23.4.5.6 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.

23.4.5.7 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](#)

23.4.5.8 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 LPI2C_MasterGetStatusFlags() .

See Also

[_lpi2c_master_flags](#).

23.4.5.9 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 <code>_lpi2c_master_flags</code> for the set of constants that should be OR'd together to form the bit mask.

23.4.5.10 static void LPI2C_MasterDisableInterrupts (`LPI2C_Type * base`, `uint32_t interruptMask`) [inline], [static]

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be disabled as interrupts.

Parameters

<i>base</i>	The LPI2C peripheral base address.
<i>interruptMask</i>	Bit mask of interrupts to disable. See <code>_lpi2c_master_flags</code> for the set of constants that should be OR'd together to form the bit mask.

23.4.5.11 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.

23.4.5.12 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.

23.4.5.13 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.

23.4.5.14 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.

23.4.5.15 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 kLPI2C_MasterTxReadyFlag 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 kLPI2C_MasterRxReadyFlag 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.

23.4.5.16 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.

23.4.5.17 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.

23.4.5.18 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.

23.4.5.19 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 kLPI2C_Read or kLPI2C_Write . 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.

23.4.5.20 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 kLPI2C_Read or kLPI2C_Write . 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.

23.4.5.21 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.

23.4.5.22 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.

23.4.5.23 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.

23.4.5.24 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.

**23.4.5.25 void LPI2C_MasterTransferCreateHandle (*LPI2C_Type* * *base*,
Ipi2c_master_handle_t * *handle*, *Ipi2c_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.

**23.4.5.26 status_t LPI2C_MasterTransferNonBlocking (*LPI2C_Type* * *base*,
Ipi2c_master_handle_t * *handle*, *Ipi2c_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.

**23.4.5.27 status_t LPI2C_MasterTransferGetCount (*LPI2C_Type* * *base*,
Ipi2c_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_NoTransferIn-Progress</i>	There is not a non-blocking transaction currently in progress.

23.4.5.28 void LPI2C_MasterTransferAbort (LPI2C_Type * *base*, Ipi2c_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.

23.4.5.29 void LPI2C_MasterTransferHandleIRQ (LPI2C_Type * *base*, void * *Ipi2cMasterHandle*)

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>lpi2cMasterHandle</i>	Pointer to the LPI2C master driver handle.

23.5 LPI2C Slave Driver

23.5.1 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 enableRx, 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 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.

23.5.2 Data Structure Documentation

23.5.2.1 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**

23.5.2.2 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`

23.5.2.3 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`

23.5.3 Typedef Documentation

23.5.3.1 **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.

23.5.4 Enumeration Type Documentation**23.5.4.1 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.

23.5.4.2 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.

23.5.4.3 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.

23.5.5 Function Documentation

23.5.5.1 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

<i>out</i>	<i>slaveConfig</i>	User provided configuration structure that is set to default values. Refer to lpi2c_slave_config_t .
------------	--------------------	--

23.5.5.2 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 LPI2C_SlaveGetDefaultConfig() 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.

23.5.5.3 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.
-------------	------------------------------------

23.5.5.4 static void LPI2C_SlaveReset (LPI2C_Type * *base*) [inline], [static]

Parameters

<i>base</i>	The LPI2C peripheral base address.
-------------	------------------------------------

23.5.5.5 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.

23.5.5.6 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](#)

23.5.5.7 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 _lpi2c_slave_flags enumerators OR'd together. You may pass the result of a previous call to LPI2C_SlaveGetStatusFlags() .

See Also

[_lpi2c_slave_flags](#).

23.5.5.8 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 _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

23.5.5.9 static void LPI2C_SlaveDisableInterrupts (LPI2C_Type * *base*, uint32_t *interruptMask*) [inline], [static]

All flags except [kLPI2C_SlaveBusyFlag](#) and [kLPI2C_SlaveBusBusyFlag](#) can be disabled as interrupts.

Parameters

<i>base</i>	The LPI2C peripheral base address.
<i>interruptMask</i>	Bit mask of interrupts to disable. See _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

23.5.5.10 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.

23.5.5.11 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.

23.5.5.12 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.

23.5.5.13 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.

23.5.5.14 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.

23.5.5.15 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.

23.5.5.16 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.

23.5.5.17 void LPI2C_SlaveTransferCreateHandle (LPI2C_Type * *base*, Ipi2c_slave_handle_t * *handle*, Ipi2c_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.

23.5.5.18 status_t LPI2C_SlaveTransferNonBlocking (LPI2C_Type * *base*, Ipi2c_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 [ipi2c_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

<code>kStatus_Success</code>	Slave transfers were successfully started.
<code>kStatus_LPI2C_Busy</code>	Slave transfers have already been started on this handle.

23.5.5.19 `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.
<i>out</i>	<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

<code>kStatus_Success</code>	
<code>kStatus_NoTransferIn-Progress</code>	

23.5.5.20 `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 lpi2c_slave_handle_t structure which stores the transfer state.

Return values

<i>kStatus_Success</i>	
<i>kStatus_LPI2C_Idle</i>	

23.5.5.21 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 lpi2c_slave_handle_t structure which stores the transfer state.

23.6 LPI2C Master DMA Driver

23.6.1 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.

23.6.2 Data Structure Documentation

23.6.2.1 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.

23.6.3 Typedef Documentation

**23.6.3.1 `typedef void(* Ipi2c_master_edma_transfer_callback_t)(LPI2C_Type *base,
Ipi2c_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.

23.6.4 Function Documentation

23.6.4.1 void LPI2C_MasterCreateEDMAHandle (LPI2C_Type * *base*, Ipi2c_master_edma_handle_t * *handle*, edma_handle_t * *rxDmaHandle*, edma_handle_t * *txDmaHandle*, Ipi2c_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_MasterTransferAbort-EDMA\(\)](#) 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.

23.6.4.2 status_t LPI2C_MasterTransferEDMA (LPI2C_Type * *base*, Ipi2c_master_edma_handle_t * *handle*, Ipi2c_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.

23.6.4.3 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.
<i>out</i>	<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.

23.6.4.4 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.

23.7 LPI2C FreeRTOS Driver

23.7.1 Overview

Driver version

- #define **FSL_LPI2C_FREERTOS_DRIVER_VERSION** (MAKE_VERSION(2, 3, 0))
LPI2C FreeRTOS driver version.

LPI2C RTOS Operation

- **status_t LPI2C_RRTOS_Init** (lpi2c_rtos_handle_t *handle, LPI2C_Type *base, const lpi2c_master_config_t *masterConfig, uint32_t srcClock_Hz)
Initializes LPI2C.
- **status_t LPI2C_RRTOS_Deinit** (lpi2c_rtos_handle_t *handle)
Deinitializes the LPI2C.
- **status_t LPI2C_RRTOS_Transfer** (lpi2c_rtos_handle_t *handle, lpi2c_master_transfer_t *transfer)
Performs I2C transfer.

23.7.2 Macro Definition Documentation

23.7.2.1 #define FSL_LPI2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

23.7.3 Function Documentation

23.7.3.1 **status_t LPI2C_RRTOS_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.

23.7.3.2 status_t LPI2C_RTOS_Deinit (*Ipi2c_rtos_handle_t * handle*)

This function deinitializes the LPI2C module and related RTOS context.

Parameters

<i>handle</i>	The RTOS LPI2C handle.
---------------	------------------------

23.7.3.3 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.

23.8 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.

23.8.1 LPI2C CMSIS Driver

23.8.1.1 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;
```

23.8.1.2 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;

```

23.8.1.3 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 24

LPSPI: Low Power Serial Peripheral Interface

24.1 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](#)

24.2 LPSPI Peripheral driver

24.2.1 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.

24.2.2 Function groups

24.2.2.1 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.

24.2.2.2 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.

24.2.2.3 LPSPI Transfer Operation

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

24.2.2.4 LPSPI Status Operation

This function group gets/clears the LPSPI status.

24.2.2.5 LPSPI Block Transfer Operation

This function group transfers a block of data, gets the transfer status, and aborts the transfer.

24.2.3 Typical use case

24.2.3.1 Master Operation

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

24.2.3.2 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_MBFI_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 }
 - enum lpspi_master_slave_mode_t {

kLPSPI_Master = 1U,

kLPSPI_Slave = 0U }
 - 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, 2, 1))
 *LPSPI driver version.*

## Initialization and deinitialization

- void [LPSPI\\_MasterInit](#) (LPSPI\_Type \*base, const [lpspi\\_master\\_config\\_t](#) \*masterConfig, uint32\_t srcClock\_Hz)

- **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 LPSPIGetInstance (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\\_SetPCSContinous](#) (LPSPI\_Type \*base, bool IsContinous)  
*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.*

## 24.2.4 Data Structure Documentation

### 24.2.4.1 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).*

## 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](#)

## 24.2.4.2 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`

#### 24.2.4.3 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.

#### 24.2.4.4 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::writeTcrInIsr**
- (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

#### 24.2.4.5 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**

## 24.2.5 Macro Definition Documentation

**24.2.5.1 #define FSL\_LPSPI\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 1))**

**24.2.5.2 #define LPSPI\_DUMMY\_DATA (0x00U)**

Dummy data used for tx if there is not txData.

**24.2.5.3 #define SPI\_RETRY\_TIMES 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/**

**24.2.5.4 #define LPSPI\_MASTER\_PCS\_SHIFT (4U)**

**24.2.5.5 #define LPSPI\_MASTER\_PCS\_MASK (0xF0U)**

**24.2.5.6 #define LPSPI\_SLAVE\_PCS\_SHIFT (4U)**

**24.2.5.7 #define LPSPI\_SLAVE\_PCS\_MASK (0xF0U)**

## 24.2.6 Typedef Documentation

**24.2.6.1 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.      |

**24.2.6.2 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.      |

## 24.2.7 Enumeration Type Documentation

### 24.2.7.1 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.

### 24.2.7.2 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.

### 24.2.7.3 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.

#### 24.2.7.4 enum \_lpspi\_dma\_enable

Enumerator

*kLPSPI\_TxDmaEnable* Transmit data DMA enable.  
*kLPSPI\_RxDmaEnable* Receive data DMA enable.

#### 24.2.7.5 enum lpspi\_master\_slave\_mode\_t

Enumerator

*kLPSPI\_Master* LPSPI peripheral operates in master mode.  
*kLPSPI\_Slave* LPSPI peripheral operates in slave mode.

#### 24.2.7.6 enum lpspi\_which\_pcs\_t

Enumerator

*kLPSPI\_Pcs0* PCS[0].  
*kLPSPI\_Pcs1* PCS[1].  
*kLPSPI\_Pcs2* PCS[2].  
*kLPSPI\_Pcs3* PCS[3].

#### 24.2.7.7 enum lpspi\_pcs\_polarity\_config\_t

Enumerator

*kLPSPI\_PcsActiveHigh* PCS Active High (idles low)  
*kLPSPI\_PcsActiveLow* PCS Active Low (idles high)

#### 24.2.7.8 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).

#### 24.2.7.9 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)

#### 24.2.7.10 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.

#### 24.2.7.11 enum lpspi\_shift\_direction\_t

Enumerator

*kLPSPI\_MsbFirst* Data transfers start with most significant bit.

*kLPSPI\_LsbFirst* Data transfers start with least significant bit.

#### 24.2.7.12 enum lpspi\_host\_request\_select\_t

Enumerator

*kLPSPI\_HostReqExtPin* Host Request is an ext pin.

*kLPSPI\_HostReqInternalTrigger* Host Request is an internal trigger.

#### 24.2.7.13 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.

#### 24.2.7.14 enum lpspi\_pin\_config\_t

Enumerator

*kLPSPISdiInSdoOut* LPSPI SDI input, SDO output.  
*kLPSPISdiInSdiOut* LPSPI SDI input, SDI output.  
*kLPSPISdoInSdoOut* LPSPI SDO input, SDO output.  
*kLPSPISdoInSdiOut* LPSPI SDO input, SDI output.

#### 24.2.7.15 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.

#### 24.2.7.16 enum lpspi\_transfer\_width\_t

Enumerator

*kLPSPISingleBitXfer* 1-bit shift at a time, data out on SDO, in on SDI (normal mode)  
*kLPSPITwoBitXfer* 2-bits shift out on SDO/SDI and in on SDO/SDI  
*kLPSPIFourBitXfer* 4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

#### 24.2.7.17 enum lpspi\_delay\_type\_t

Enumerator

*kLPSPIPcsToSck* PCS-to-SCK delay.  
*kLPSPILastSckToPcs* Last SCK edge to PCS delay.  
*kLPSPIBetweenTransfer* Delay between transfers.

#### 24.2.7.18 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.

#### 24.2.7.19 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.
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\_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI\_SlaveByteSwap flag.

#### 24.2.7.20 enum \_lpspi\_transfer\_state

Enumerator

***kLPSPI\_Idle*** Nothing in the transmitter/receiver.

**kLPSPI\_Busy** Transfer queue is not finished.

**kLPSPI\_Error** Transfer error.

## 24.2.8 Function Documentation

### 24.2.8.1 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                           |

### 24.2.8.2 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 |
|---------------------|------------------------------------------------------------|

### 24.2.8.3 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> . |

### 24.2.8.4 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. |
|--------------------|------------------------------------------------------------|

#### 24.2.8.5 void LPSPI\_Deinit ( LPSPI\_Type \* *base* )

Call this API to disable the LPSPI clock.

Parameters

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

#### 24.2.8.6 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. |
|-------------|---------------------------|

#### 24.2.8.7 uint32\_t LPSPIGetInstance ( LPSPI\_Type \* *base* )

Parameters

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

Returns

LPSPI instance.

#### 24.2.8.8 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. |

#### 24.2.8.9 static uint32\_t LPSPI\_GetStatusFlags ( LPSPI\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The LPSPI status(in SR register).

#### 24.2.8.10 static uint8\_t LPSPI\_GetTxFifoSize ( LPSPI\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The LPSPI Tx FIFO size.

#### 24.2.8.11 static uint8\_t LPSPI\_GetRxFifoSize ( LPSPI\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The LPSPI Rx FIFO size.

#### 24.2.8.12 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.

#### 24.2.8.13 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.

#### 24.2.8.14 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 _lpspi_flags. |

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

#### 24.2.8.15 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 _lpspi_interrupt_enable. |

#### 24.2.8.16 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 _lpspi_interrupt_enable. |

#### 24.2.8.17 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 _lpspi_dma_enable. |

#### 24.2.8.18 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 _lpspi_dma_enable. |

#### 24.2.8.19 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.

#### 24.2.8.20 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.

#### 24.2.8.21 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.

#### 24.2.8.22 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. |

#### 24.2.8.23 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. |

#### 24.2.8.24 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>IsContinous</i> | True to set the transfer PCS to continuous mode, false to set to uncontinuous mode. |

#### 24.2.8.25 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.

#### 24.2.8.26 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. |

#### 24.2.8.27 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. |

#### 24.2.8.28 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 _lpspi_pcs_polarity. |

#### 24.2.8.29 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. |

#### 24.2.8.30 `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.

#### 24.2.8.31 `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. |

#### 24.2.8.32 `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 `delayTime = LPSPI_clockSource / (PRESCALE * Delay_scaler)`.

Parameters

|                                  |                                                                                             |
|----------------------------------|---------------------------------------------------------------------------------------------|
| <code>base</code>                | LPSPI peripheral address.                                                                   |
| <code>delayTimeIn-NanoSec</code> | The desired delay value in nano-seconds.                                                    |
| <code>whichDelay</code>          | The desired delay to configuration, which must be of type <code>lpspi_delay_type_t</code> . |
| <code>srcClock_Hz</code>         | Module source input clock in Hertz.                                                         |

Returns

actual Calculated delay value in nano-seconds.

#### 24.2.8.33 `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

|                   |                           |
|-------------------|---------------------------|
| <code>base</code> | LPSPI peripheral address. |
| <code>data</code> | The data word to be sent. |

**24.2.8.34 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.

#### 24.2.8.35 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. |

#### 24.2.8.36 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.                   |

#### 24.2.8.37 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.

#### 24.2.8.38 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 lpspi_master_handle_t structure which stores the transfer state. |
| <i>transfer</i> | pointer to <a href="#">lpspi_transfer_t</a> structure.                      |

Returns

status of status\_t.

#### 24.2.8.39 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`.

#### 24.2.8.40 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. |

#### 24.2.8.41 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. |

#### 24.2.8.42 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.                                |

#### 24.2.8.43 `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`.

#### 24.2.8.44 `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`.

**24.2.8.45 void LPSPI\_SlaveTransferAbort ( LPSPI\_Type \* *base*, Ispspi\_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 lpspi_slave_handle_t structure which stores the transfer state. |

#### 24.2.8.46 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 lpspi_slave_handle_t structure which stores the transfer state. |

### 24.2.9 Variable Documentation

#### 24.2.9.1 volatile uint8\_t g\_lpspiDummyData[]

## 24.3 LPSPI eDMA Driver

### 24.3.1 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.*
- 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, 1, 0))  
*LPSPI EDMA driver version.*

### 24.3.2 Data Structure Documentation

#### 24.3.2.1 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`

#### 24.3.2.2 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`

### 24.3.3 Macro Definition Documentation

**24.3.3.1 `#define FSL_LPSPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`**

### 24.3.4 Typedef Documentation

**24.3.4.1 `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.   |

**24.3.4.2 `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.   |

### 24.3.5 Function Documentation

```
24.3.5.1 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 Rx 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 edmaIntermediaryTo-TxRegHandle. (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>                    | LP SPI handle pointer to lpspi_master_edma_handle_t.               |
| <i>callback</i>                  | LP SPI callback.                                                   |
| <i>userData</i>                  | callback function parameter.                                       |
| <i>edmaRxRegTo-RxDataHandle</i>  | edmaRxRegToRxDataHandle pointer to <a href="#">edma_handle_t</a> . |
| <i>edmaTxData-ToTxReg-Handle</i> | edmaTxDataToTxRegHandle pointer to <a href="#">edma_handle_t</a> . |

```
24.3.5.2 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 <code>lpspi_transfer_t</code> structure.                                           |

Returns

status of `status_t`.

#### 24.3.5.3 `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 <code>lpspi_master_edma_handle_t</code> structure which stores the transfer state. |

#### 24.3.5.4 `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 <code>lpspi_master_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`.

#### 24.3.5.5 `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 edmaRxRegToRxDataHandle 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 lpspi_slave_edma_handle_t.                 |
| <i>callback</i>                | LPSPI callback.                                                    |
| <i>userData</i>                | callback function parameter.                                       |
| <i>edmaRxRegToRxDataHandle</i> | edmaRxRegToRxDataHandle pointer to <a href="#">edma_handle_t</a> . |
| <i>edmaTxDataToTxRegHandle</i> | edmaTxDataToTxRegHandle pointer to <a href="#">edma_handle_t</a> . |

#### 24.3.5.6 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 lpspi_slave_edma_handle_t structure which stores the transfer state. |
| <i>transfer</i> | pointer to <a href="#">lpspi_transfer_t</a> structure.                          |

Returns

status of status\_t.

#### 24.3.5.7 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 lpspi_slave_edma_handle_t structure which stores the transfer state. |

#### 24.3.5.8 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 lpspi_slave_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.

## 24.4 LPSPI FreeRTOS Driver

### 24.4.1 Overview

#### Driver version

- #define `FSL_LPSPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))`  
*LPSPI FreeRTOS driver version 2.0.5.*

#### 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.*

### 24.4.2 Macro Definition Documentation

#### 24.4.2.1 #define `FSL_LPSPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))`

### 24.4.3 Function Documentation

#### 24.4.3.1 `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

|                           |                                                                            |
|---------------------------|----------------------------------------------------------------------------|
| <code>handle</code>       | The RTOS LPSPI handle, the pointer to an allocated space for RTOS context. |
| <code>base</code>         | The pointer base address of the LPSPI instance to initialize.              |
| <code>masterConfig</code> | Configuration structure to set-up LPSPI in master mode.                    |
| <code>srcClock_Hz</code>  | Frequency of input clock of the LPSPI module.                              |

Returns

status of the operation.

#### 24.4.3.2 status\_t LPSPI\_RTOS\_Deinit ( *Ipspi\_rtos\_handle\_t \* handle* )

This function deinitializes the LPSPI module and related RTOS context.

Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | The RTOS LPSPI handle. |
|---------------|------------------------|

#### 24.4.3.3 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.

## 24.5 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>.

### 24.5.1 Function groups

#### 24.5.1.1 LPSPI CMSIS GetVersion Operation

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

#### 24.5.1.2 LPSPI CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

#### 24.5.1.3 LPSPI CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialized the instance in master mode or slave mode. And this API must be called before you configure an instance or after you Deinit an instance. The right steps to start an instance is that you must initialize the instance which been selected firstly, then you can power on the instance. After these all have been done, you can configure the instance by using control operation. If you want to Uninitialize the instance, you must power off the instance first.

#### 24.5.1.4 LPSPI Transfer Operation

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

#### 24.5.1.5 LPSPI Status Operation

This function group gets the LPSPI transfer status.

#### 24.5.1.6 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.

## 24.5.2 Typical use case

### 24.5.2.1 Master Operation

```
/* Variables */
uint8_t masterRxData[TRANSFER_SIZE] = {0U};
uint8_t masterTxData[TRANSFER_SIZE] = {0U};

/*DSPI master init*/
Driver_SPI0.Initialize(DSPI_MasterSignalEvent_t);
Driver_SPI0.PowerControl(ARM_POWER_FULL);
Driver_SPI0.Control(ARM_SPI_MODE_MASTER, TRANSFER_BAUDRATE);

/* Start master transfer */
Driver_SPI0.Transfer(masterTxData, masterRxData, TRANSFER_SIZE);

/* Master power off */
Driver_SPI0.PowerControl(ARM_POWER_OFF);

/* Master uninitialize */
Driver_SPI0.Uninitialize();
```

### 24.5.2.2 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 uninitialize */
Driver_SPI2.Uninitialize();
```

## Chapter 25

# LPUART: Low Power Universal Asynchronous Receiver-/Transmitter Driver

### 25.1 Overview

#### Modules

- [LPUART CMSIS Driver](#)
- [LPUART Driver](#)
- [LPUART FreeRTOS Driver](#)
- [LPUART eDMA Driver](#)

## 25.2 LPUART Driver

### 25.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power UART (LPUART) module of MCUXpresso SDK devices.

### 25.2.2 Typical use case

#### 25.2.2.1 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,
   
kLPUART\_SevenDataBits = 0x1U }
- LPUART data bits count.*
- enum `lpuart_stop_bit_count_t` {
   
kLPUART\_OneStopBit = 0U,
   
kLPUART\_TwoStopBit = 1U }
- LPUART stop bit count.*
- enum `lpuart_transmit_cts_source_t` {
   
kLPUART\_CtsSourcePin = 0U,
   
kLPUART\_CtsSourceMatchResult = 1U }
- LPUART transmit CTS source.*
- enum `lpuart_transmit_cts_config_t` {
   
kLPUART\_CtsSampleAtStart = 0U,
   
kLPUART\_CtsSampleAtIdle = 1U }
- LPUART transmit CTS configure.*
- enum `lpuart_idle_type_select_t` {
   
kLPUART\_IdleTypeStartBit = 0U,
   
kLPUART\_IdleTypeStopBit = 1U }
- LPUART idle flag type defines when the receiver starts counting.*
- enum `lpuart_idle_config_t` {

```
kLPUART_IdleCharacter1 = 0U,
kLPUART_IdleCharacter2 = 1U,
kLPUART_IdleCharacter4 = 2U,
kLPUART_IdleCharacter8 = 3U,
kLPUART_IdleCharacter16 = 4U,
kLPUART_IdleCharacter32 = 5U,
kLPUART_IdleCharacter64 = 6U,
kLPUART_IdleCharacter128 = 7U }
```

*LPUART idle detected configuration.*

- enum \_lpuart\_interrupt\_enable {

```
kLPUART_LinBreakInterruptEnable = (LPUART_BAUD_LBKDIIE_MASK >> 8U),
kLPUART_RxActiveEdgeInterruptEnable = (LPUART_BAUD_RXEDGIE_MASK >> 8U),
kLPUART_TxDataRegEmptyInterruptEnable = (LPUART_CTRL_TIE_MASK),
kLPUART_TransmissionCompleteInterruptEnable = (LPUART_CTRL_TCIE_MASK),
kLPUART_RxDataRegFullInterruptEnable = (LPUART_CTRL_RIE_MASK),
kLPUART_IdleLineInterruptEnable = (LPUART_CTRL_ILIE_MASK),
kLPUART_RxOverrunInterruptEnable = (LPUART_CTRL_ORIE_MASK),
kLPUART_NoiseErrorInterruptEnable = (LPUART_CTRL_NEIE_MASK),
kLPUART_FramingErrorInterruptEnable = (LPUART_CTRL_FEIE_MASK),
kLPUART_ParityErrorInterruptEnable = (LPUART_CTRL_PEIE_MASK),
kLPUART_Match1InterruptEnable = (LPUART_CTRL_MA1IE_MASK),
kLPUART_Match2InterruptEnable = (LPUART_CTRL_MA2IE_MASK),
kLPUART_TxFifoOverflowInterruptEnable = (LPUART_FIFO_TXOFE_MASK),
kLPUART_RxFifoUnderflowInterruptEnable = (LPUART_FIFO_RXUFE_MASK) }
```

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

- enum \_lpuart\_flags {

```
kLPUART_TxDataRegEmptyFlag,
kLPUART_TransmissionCompleteFlag,
kLPUART_RxDataRegFullFlag = (LPUART_STAT_RDRF_MASK),
kLPUART_IdleLineFlag = (LPUART_STAT_IDLE_MASK),
kLPUART_RxOverrunFlag = (LPUART_STAT_OR_MASK),
kLPUART_NoiseErrorFlag = (LPUART_STAT_NF_MASK),
kLPUART_FramingErrorFlag,
kLPUART_ParityErrorFlag = (LPUART_STAT_PF_MASK),
kLPUART_LinBreakFlag = (LPUART_STAT_LBKDIF_MASK),
kLPUART_RxActiveEdgeFlag = (LPUART_STAT_RXEDGIF_MASK),
kLPUART_RxActiveFlag,
kLPUART_DataMatch1Flag,
kLPUART_DataMatch2Flag,
kLPUART_TxFifoEmptyFlag,
kLPUART_RxFifoEmptyFlag,
kLPUART_TxFifoOverflowFlag,
kLPUART_RxFifoUnderflowFlag }
```

*LPUART status flags.*

## Driver version

- `#define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 5, 2))`  
*LPUART driver version.*

## Software Reset

- static void `LPUART_SoftwareReset` (LPUART\_Type \*base)  
*Resets the LPUART using software.*

## 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.*
- static void `LPUART_SetRxFifoWatermark` (LPUART\_Type \*base, uint8\_t water)  
*Sets the rx FIFO watermark.*
- static void `LPUART_SetTxFifoWatermark` (LPUART\_Type \*base, uint8\_t water)  
*Sets the tx FIFO watermark.*

## Status

- `uint32_t LPUART_GetStatusFlags` (LPUART\_Type \*base)  
*Gets LPUART status flags.*
- `status_t LPUART_ClearStatusFlags` (LPUART\_Type \*base, uint32\_t mask)  
*Clears status flags with a provided mask.*

## Interrupts

- void [LPUART\\_EnableInterrupts](#) (LPUART\_Type \*base, uint32\_t mask)  
*Enables LPUART interrupts according to a provided mask.*
- void [LPUART\\_DisableInterrupts](#) (LPUART\_Type \*base, uint32\_t mask)  
*Disables LPUART interrupts according to a provided mask.*
- uint32\_t [LPUART\\_GetEnabledInterrupts](#) (LPUART\_Type \*base)  
*Gets enabled LPUART interrupts.*

## DMA Configuration

- static uint32\_t [LPUART\\_GetDataRegisterAddress](#) (LPUART\_Type \*base)  
*Gets the LPUART data register address.*
- static void [LPUART\\_EnableTxDMA](#) (LPUART\_Type \*base, bool enable)  
*Enables or disables the LPUART transmitter DMA request.*
- static void [LPUART\\_EnableRxDMA](#) (LPUART\_Type \*base, bool enable)  
*Enables or disables the LPUART receiver DMA.*

## Bus Operations

- uint32\_t [LPUARTGetInstance](#) (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.*
- static uint8\_t [LPUART\\_GetRxFifoCount](#) (LPUART\_Type \*base)  
*Gets the rx FIFO data count.*
- static uint8\_t [LPUART\\_GetTxFifoCount](#) (LPUART\_Type \*base)  
*Gets the tx FIFO data count.*
- 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.*

## 25.2.3 Data Structure Documentation

### 25.2.3.1 struct lpuart\_config\_t

#### Data Fields

- `uint32_t baudRate_Bps`  
*LPUART baud rate.*
- `lpuart_parity_mode_t parityMode`  
*Parity mode, disabled (default), even, odd.*
- `lpuart_data_bits_t dataBitsCount`  
*Data bits count, eight (default), seven.*
- `bool isMsb`  
*Data bits order, LSB (default), MSB.*
- `lpuart_stop_bit_count_t stopBitCount`  
*Number of stop bits, 1 stop bit (default) or 2 stop bits.*
- `uint8_t txFifoWatermark`  
*TX FIFO watermark.*
- `uint8_t rxFifoWatermark`

- `bool enableRxRTS`  
*RX RTS enable.*
- `bool enableTxCTS`  
*TX CTS enable.*
- `lpuart_transmit_cts_source_t txCtsSource`  
*TX CTS source.*
- `lpuart_transmit_cts_config_t txCtsConfig`  
*TX CTS configure.*
- `lpuart_idle_type_select_t rxIdleType`  
*RX IDLE type.*
- `lpuart_idle_config_t rxIdleConfig`  
*RX IDLE configuration.*
- `bool enableTx`  
*Enable TX.*
- `bool enableRx`  
*Enable RX.*

## Field Documentation

(1) `lpuart_idle_type_select_t lpuart_config_t::rxIdleType`

(2) `lpuart_idle_config_t lpuart_config_t::rxIdleConfig`

### 25.2.3.2 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`

### 25.2.3.3 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.*
- bool **isSevenDataBits**  
*Seven data bits flag.*

#### 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`
- (15) `bool lpuart_handle_t::isSevenDataBits`

## 25.2.4 Macro Definition Documentation

**25.2.4.1 #define FSL\_LPUART\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 2))**

**25.2.4.2 #define UART\_RETRY\_TIMES 0U /\* Defining to zero means to keep waiting for the flag until it is assert/deassert. \*/**

## 25.2.5 Typedef Documentation

**25.2.5.1 typedef void(\* lpuart\_transfer\_callback\_t)(LPUART\_Type \*base, lpuart\_handle\_t \*handle, status\_t status, void \*userData)**

## 25.2.6 Enumeration Type Documentation

### 25.2.6.1 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.

### 25.2.6.2 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.

### 25.2.6.3 enum lpuart\_data\_bits\_t

Enumerator

*kLPUART\_EightDataBits* Eight data bit.

*kLPUART\_SevenDataBits* Seven data bit.

### 25.2.6.4 enum lpuart\_stop\_bit\_count\_t

Enumerator

*kLPUART\_OneStopBit* One stop bit.

*kLPUART\_TwoStopBit* Two stop bits.

### 25.2.6.5 enum lpuart\_transmit\_cts\_source\_t

Enumerator

*kLPUART\_CtsSourcePin* CTS resource is the LPUART\_CTS pin.

*kLPUART\_CtsSourceMatchResult* CTS resource is the match result.

### 25.2.6.6 enum lpuart\_transmit\_cts\_config\_t

Enumerator

*kLPUART\_CtsSampleAtStart* CTS input is sampled at the start of each character.

*kLPUART\_CtsSampleAtIdle* CTS input is sampled when the transmitter is idle.

### 25.2.6.7 enum lpuart\_idle\_type\_select\_t

Enumerator

***kLPUART\_IdleTypeStartBit*** Start counting after a valid start bit.

***kLPUART\_IdleTypeStopBit*** Start counting after a stop bit.

### 25.2.6.8 enum lpuart\_idle\_config\_t

This structure defines the number of idle characters that must be received before the IDLE flag is set.

Enumerator

***kLPUART\_IdleCharacter1*** the number of idle characters.

***kLPUART\_IdleCharacter2*** the number of idle characters.

***kLPUART\_IdleCharacter4*** the number of idle characters.

***kLPUART\_IdleCharacter8*** the number of idle characters.

***kLPUART\_IdleCharacter16*** the number of idle characters.

***kLPUART\_IdleCharacter32*** the number of idle characters.

***kLPUART\_IdleCharacter64*** the number of idle characters.

***kLPUART\_IdleCharacter128*** the number of idle characters.

### 25.2.6.9 enum \_lpuart\_interrupt\_enable

This structure contains the settings for all LPUART interrupt configurations.

Enumerator

***kLPUART\_LinBreakInterruptEnable*** LIN break detect. bit 7

***kLPUART\_RxActiveEdgeInterruptEnable*** Receive Active Edge. bit 6

***kLPUART\_TxDataRegEmptyInterruptEnable*** Transmit data register empty. bit 23

***kLPUART\_TransmissionCompleteInterruptEnable*** Transmission complete. bit 22

***kLPUART\_RxDataRegFullInterruptEnable*** Receiver data register full. bit 21

***kLPUART\_IdleLineInterruptEnable*** Idle line. bit 20

***kLPUART\_RxOverrunInterruptEnable*** Receiver Overrun. bit 27

***kLPUART\_NoiseErrorInterruptEnable*** Noise error flag. bit 26

***kLPUART\_FramingErrorInterruptEnable*** Framing error flag. bit 25

***kLPUART\_ParityErrorInterruptEnable*** Parity error flag. bit 24

***kLPUART\_Match1InterruptEnable*** Parity error flag. bit 15

***kLPUART\_Match2InterruptEnable*** Parity error flag. bit 14

***kLPUART\_TxFifoOverflowInterruptEnable*** Transmit FIFO Overflow. bit 9

***kLPUART\_RxFifoUnderflowInterruptEnable*** Receive FIFO Underflow. bit 8

### 25.2.6.10 enum \_lpuart\_flags

This provides constants for the LPUART status flags for use in the LPUART functions.

Enumerator

***kLPUART\_TxDataRegEmptyFlag*** Transmit data register empty flag, sets when transmit buffer is empty. bit 23

***kLPUART\_TransmissionCompleteFlag*** Transmission complete flag, sets when transmission activity complete. bit 22

***kLPUART\_RxDataRegFullFlag*** Receive data register full flag, sets when the receive data buffer is full. bit 21

***kLPUART\_IdleLineFlag*** Idle line detect flag, sets when idle line detected. bit 20

***kLPUART\_RxOverrunFlag*** Receive Overrun, sets when new data is received before data is read from receive register. bit 19

***kLPUART\_NoiseErrorFlag*** Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18

***kLPUART\_FramingErrorFlag*** Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17

***kLPUART\_ParityErrorFlag*** If parity enabled, sets upon parity error detection. bit 16

***kLPUART\_LinBreakFlag*** LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31

***kLPUART\_RxActiveEdgeFlag*** Receive pin active edge interrupt flag, sets when active edge detected. bit 30

***kLPUART\_RxActiveFlag*** Receiver Active Flag (RAF), sets at beginning of valid start. bit 24

***kLPUART\_DataMatch1Flag*** The next character to be read from LPUART\_DATA matches MA1. bit 15

***kLPUART\_DataMatch2Flag*** The next character to be read from LPUART\_DATA matches MA2. bit 14

***kLPUART\_TxFifoEmptyFlag*** TXEMPT bit, sets if transmit buffer is empty. bit 7

***kLPUART\_RxFifoEmptyFlag*** RXEMPT bit, sets if receive buffer is empty. bit 6

***kLPUART\_TxFifoOverflowFlag*** TXOF bit, sets if transmit buffer overflow occurred. bit 1

***kLPUART\_RxFifoUnderflowFlag*** RXUF bit, sets if receive buffer underflow occurred. bit 0

### 25.2.7 Function Documentation

#### 25.2.7.1 static void LPUART\_SoftwareReset ( **LPUART\_Type** \* *base* ) [inline], [static]

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

## Parameters

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

### 25.2.7.2 status\_t LPUART\_Init ( LPUART\_Type \* *base*, const lpuart\_config\_t \* *config*, uint32\_t *srcClock\_Hz* )

This function configures the LPUART module with user-defined settings. Call the [LPUART\\_GetDefaultConfig\(\)](#) function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
* lpuart_config_t lpuartConfig;
* lpuartConfig.baudRate_Bps = 115200U;
* lpuartConfig.parityMode = kLPUART_ParityDisabled;
* lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
* lpuartConfig.isMsb = false;
* lpuartConfig.stopBitCount = kLPUART_OneStopBit;
* lpuartConfig.txFifoWatermark = 0;
* lpuartConfig.rxFifoWatermark = 1;
* LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
*
```

## Parameters

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>base</i>        | LPUART peripheral base address.                    |
| <i>config</i>      | Pointer to a user-defined configuration structure. |
| <i>srcClock_Hz</i> | LPUART clock source frequency in HZ.               |

## Return values

|                                          |                                                  |
|------------------------------------------|--------------------------------------------------|
| <i>kStatus_LPUART_BaudrateNotSupport</i> | Baudrate is not support in current clock source. |
| <i>kStatus_Success</i>                   | LPUART initialize succeed                        |

### 25.2.7.3 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. |
|-------------|---------------------------------|

#### 25.2.7.4 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. |
|---------------|---------------------------------------|

#### 25.2.7.5 status\_t LPUART\_SetBaudRate ( LPUART\_Type \* *base*, uint32\_t *baudRate\_Bps*, uint32\_t *srcClock\_Hz* )

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART\_Init.

```
* LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
*
```

Parameters

|                     |                                      |
|---------------------|--------------------------------------|
| <i>base</i>         | LPUART peripheral base address.      |
| <i>baudRate_Bps</i> | LPUART baudrate to be set.           |
| <i>srcClock_Hz</i>  | LPUART clock source frequency in HZ. |

Return values

|                                          |                                                        |
|------------------------------------------|--------------------------------------------------------|
| <i>kStatus_LPUART_BaudrateNotSupport</i> | Baudrate is not supported in the current clock source. |
| <i>kStatus_Success</i>                   | Set baudrate succeeded.                                |

#### 25.2.7.6 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. |

#### 25.2.7.7 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.          |

#### 25.2.7.8 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. |

#### 25.2.7.9 static void LPUART\_SetRxFifoWatermark ( LPUART\_Type \* *base*, uint8\_t *water* ) [inline], [static]

Parameters

|              |                                 |
|--------------|---------------------------------|
| <i>base</i>  | LPUART peripheral base address. |
| <i>water</i> | Rx FIFO watermark.              |

### 25.2.7.10 static void LPUART\_SetTxFifoWatermark ( LPUART\_Type \* *base*, uint8\_t *water* ) [inline], [static]

Parameters

|              |                                 |
|--------------|---------------------------------|
| <i>base</i>  | LPUART peripheral base address. |
| <i>water</i> | Tx FIFO watermark.              |

### 25.2.7.11 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](#).

### 25.2.7.12 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\_NoiseErrorInRxDataRegFlag, kLPUART\_ParityErrorInRxDataRegFlag, 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_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.                                                         |

**25.2.7.13 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 the enumeration <code>_uart_interrupt_enable</code> . |

**25.2.7.14 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> . |

### 25.2.7.15 `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](#).

### 25.2.7.16 `static uint32_t LPUART_GetDataRegisterAddress ( LPUART_Type * base ) [inline], [static]`

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

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

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

### 25.2.7.17 `static void LPUART_EnableTxDMA ( LPUART_Type * base, bool enable ) [inline], [static]`

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPUART peripheral base address.   |
| <i>enable</i> | True to enable, false to disable. |

#### 25.2.7.18 static void LPUART\_EnableRxDMA ( LPUART\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPUART peripheral base address.   |
| <i>enable</i> | True to enable, false to disable. |

#### 25.2.7.19 uint32\_t LPUART\_GetInstance ( LPUART\_Type \* *base* )

Parameters

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

Returns

LPUART instance.

#### 25.2.7.20 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. |

#### 25.2.7.21 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. |

### 25.2.7.22 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.  |

### 25.2.7.23 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.

### 25.2.7.24 static uint8\_t LPUART\_GetRx\_fifoCount ( LPUART\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

rx FIFO data count.

#### 25.2.7.25 static uint8\_t LPUART\_GetTxFifoCount ( LPUART\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

tx FIFO data count.

#### 25.2.7.26 void LPUART\_SendAddress ( LPUART\_Type \* *base*, uint8\_t *address* )

Parameters

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

#### 25.2.7.27 status\_t LPUART\_WriteBlocking ( LPUART\_Type \* *base*, const uint8\_t \* *data*, size\_t *length* )

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the dat to be sent out to the bus.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | LPUART peripheral base address.     |
| <i>data</i>   | Start address of the data to write. |
| <i>length</i> | Size of the data to write.          |

Return values

|                                     |                                         |
|-------------------------------------|-----------------------------------------|
| <i>kStatus_LPUART_-<br/>Timeout</i> | Transmission timed out and was aborted. |
| <i>kStatus_Success</i>              | Successfully wrote all data.            |

### 25.2.7.28 **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-<br/>HardwareOverrun</i> | Receiver overrun happened while receiving data. |
| <i>kStatus_LPUART_Noise-<br/>Error</i>        | Noise error happened while receiving data.      |
| <i>kStatus_LPUART_-<br/>FramingError</i>      | Framing error happened while receiving data.    |
| <i>kStatus_LPUART_Parity-<br/>Error</i>       | Parity error happened while receiving data.     |
| <i>kStatus_LPUART_-<br/>Timeout</i>           | Transmission timed out and was aborted.         |
| <i>kStatus_Success</i>                        | Successfully received all data.                 |

### 25.2.7.29 **void LPUART\_TransferCreateHandle ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle*, Ipuart\_transfer\_callback\_t *callback*, void \* *userData* )**

This function initializes the LPUART handle, which can be used for other LPUART transactional APIs. Usually, for a specified LPUART instance, call this API once to get the initialized handle.

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the [LPUART\\_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, the user

can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as `ringBuffer`.

Parameters

|                       |                                 |
|-----------------------|---------------------------------|
| <code>base</code>     | LPUART peripheral base address. |
| <code>handle</code>   | LPUART handle pointer.          |
| <code>callback</code> | Callback function.              |
| <code>userData</code> | User data.                      |

### 25.2.7.30 `status_t LPUART_TransferSendNonBlocking ( LPUART_Type * base, Ipuart_handle_t * handle, Ipuart_transfer_t * xfer )`

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the `kStatus_LPUART_TxIdle` as status parameter.

Note

The `kStatus_LPUART_TxIdle` is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the T-X, check the `kLPUART_TransmissionCompleteFlag` to ensure that the transmit is finished.

Parameters

|                     |                                                                    |
|---------------------|--------------------------------------------------------------------|
| <code>base</code>   | LPUART peripheral base address.                                    |
| <code>handle</code> | LPUART handle pointer.                                             |
| <code>xfer</code>   | LPUART transfer structure, see <a href="#">Ipuart_transfer_t</a> . |

Return values

|                                      |                                                                                    |
|--------------------------------------|------------------------------------------------------------------------------------|
| <code>kStatus_Success</code>         | Successfully start the data transmission.                                          |
| <code>kStatus_LPUART_TxBusy</code>   | Previous transmission still not finished, data not all written to the TX register. |
| <code>kStatus_InvalidArgument</code> | Invalid argument.                                                                  |

### 25.2.7.31 `void LPUART_TransferStartRingBuffer ( LPUART_Type * base, Ipuart_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.                                                                     |

### 25.2.7.32 void LPUART\_TransferStopRingBuffer ( `LPUART_Type * base, Ipuart_handle_t * handle` )

This function aborts the background transfer and uninstalls the ring buffer.

#### Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

### 25.2.7.33 size\_t LPUART\_TransferGetRxRingBufferLength ( `LPUART_Type * base, Ipuart_handle_t * handle` )

#### Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

#### Returns

Length of received data in RX ring buffer.

**25.2.7.34 void LPUART\_TransferAbortSend ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle* )**

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are not sent out.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

### 25.2.7.35 status\_t LPUART\_TransferGetSendCount ( **LPUART\_Type \* base,**                  **Ipuart\_handle\_t \* handle, uint32\_t \* count** )

This function gets the number of bytes that have been sent out to bus by an interrupt method.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |
| <i>count</i>  | Send bytes count.               |

Return values

|                                     |                                                       |
|-------------------------------------|-------------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No send in progress.                                  |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                 |
| <i>kStatus_Success</i>              | Get successfully through the parameter <i>count</i> ; |

### 25.2.7.36 status\_t LPUART\_TransferReceiveNonBlocking ( **LPUART\_Type \* base,**                  **Ipuart\_handle\_t \* handle, Ipuart\_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.                                        |

### 25.2.7.37 void LPUART\_TransferAbortReceive ( `LPUART_Type * base, Ipuart_handle_t * handle` )

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

### 25.2.7.38 status\_t LPUART\_TransferGetReceiveCount ( `LPUART_Type * base, Ipuart_handle_t * handle, uint32_t * count` )

This function gets the number of bytes that have been received.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |
| <i>count</i>  | Receive bytes count.            |

Return values

|                                     |                                               |
|-------------------------------------|-----------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                       |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                         |
| <i>kStatus_Success</i>              | Get successfully through the parameter count; |

### 25.2.7.39 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.          |

### 25.2.7.40 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.          |

## 25.3 LPUART eDMA Driver

### 25.3.1 Overview

#### Data Structures

- struct [lpuart\\_edma\\_handle\\_t](#)  
*LPUART eDMA handle.* [More...](#)

#### TypeDefs

- [typedef void\(\\* lpuart\\_edma\\_transfer\\_callback\\_t \)](#)(LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*LPUART transfer callback function.*

#### Driver version

- #define [FSL\\_LPUART\\_EDMA\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 5, 2))  
*LPUART EDMA driver version.*

#### eDMA transactional

- void [LPUART\\_TransferCreateHandleEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, [lpuart\\_edma\\_transfer\\_callback\\_t](#) callback, void \*userData, [edma\\_handle\\_t](#) \*txEdmaHandle, [edma\\_handle\\_t](#) \*rxEdmaHandle)  
*Initializes the LPUART handle which is used in transactional functions.*
- [status\\_t LPUART\\_SendEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, [lpuart\\_transfer\\_t](#) \*xfer)  
*Sends data using eDMA.*
- [status\\_t LPUART\\_ReceiveEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, [lpuart\\_transfer\\_t](#) \*xfer)  
*Receives data using eDMA.*
- void [LPUART\\_TransferAbortSendEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle)  
*Aborts the sent data using eDMA.*
- void [LPUART\\_TransferAbortReceiveEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle)  
*Aborts the received data using eDMA.*
- [status\\_t LPUART\\_TransferGetSendCountEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of bytes written to the LPUART TX register.*
- [status\\_t LPUART\\_TransferGetReceiveCountEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of received bytes.*
- void [LPUART\\_TransferEdmaHandleIRQ](#) (LPUART\_Type \*base, void \*lpuartEdmaHandle)  
*LPUART eDMA IRQ handle function.*

## 25.3.2 Data Structure Documentation

### 25.3.2.1 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`

## 25.3.3 Macro Definition Documentation

### 25.3.3.1 #define FSL\_LPUART\_EDMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 2))

## 25.3.4 Typedef Documentation

**25.3.4.1 `typedef void(* lpuart_edma_transfer_callback_t)(LPUART_Type *base, lpuart_edma_handle_t *handle, status_t status, void *userData)`**

## 25.3.5 Function Documentation

**25.3.5.1 `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.          |

**25.3.5.2 `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.           |

### 25.3.5.3 status\_t LPUART\_ReceiveEDMA ( LPUART\_Type \* *base*, Ipuart\_edma\_handle\_t \* *handle*, Ipuart\_transfer\_t \* *xfer* )

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

|               |                                                                         |
|---------------|-------------------------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                                         |
| <i>handle</i> | Pointer to Ipuart_edma_handle_t structure.                              |
| <i>xfer</i>   | LPUART eDMA transfer structure, see <a href="#">Ipuart_transfer_t</a> . |

Return values

|                                |                            |
|--------------------------------|----------------------------|
| <i>kStatus_Success</i>         | if succeed, others fail.   |
| <i>kStatus_LPUART_Rx-Busy</i>  | Previous transfer ongoing. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.          |

### 25.3.5.4 void LPUART\_TransferAbortSendEDMA ( LPUART\_Type \* *base*, Ipuart\_edma\_handle\_t \* *handle* )

This function aborts the sent data using eDMA.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.            |
| <i>handle</i> | Pointer to Ipuart_edma_handle_t structure. |

### 25.3.5.5 void LPUART\_TransferAbortReceiveEDMA ( LPUART\_Type \* *base*, Ipuart\_edma\_handle\_t \* *handle* )

This function aborts the received data using eDMA.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.            |
| <i>handle</i> | Pointer to lpuart_edma_handle_t structure. |

### 25.3.5.6 status\_t LPUART\_TransferGetSendCountEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |
| <i>count</i>  | Send bytes count.               |

Return values

|                                     |                                                       |
|-------------------------------------|-------------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No send in progress.                                  |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                 |
| <i>kStatus_Success</i>              | Get successfully through the parameter <i>count</i> ; |

### 25.3.5.7 status\_t LPUART\_TransferGetReceiveCountEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of received bytes.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |
| <i>count</i>  | Receive bytes count.            |

Return values

|                                     |                                               |
|-------------------------------------|-----------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                       |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                         |
| <i>kStatus_Success</i>              | Get successfully through the parameter count; |

#### 25.3.5.8 void LPUART\_TransferEdmaHandleIRQ ( **LPUART\_Type** \* *base*, **void** \* *lpuartEdmaHandle* )

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

Note

This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

|                         |                                 |
|-------------------------|---------------------------------|
| <i>base</i>             | LPUART peripheral base address. |
| <i>lpuartEdmaHandle</i> | LPUART handle pointer.          |

## 25.4 LPUART FreeRTOS Driver

### 25.4.1 Overview

#### Data Structures

- struct `lpuart_rtos_config_t`  
*LPUART RTOS configuration structure.* [More...](#)

#### Driver version

- #define `FSL_LPUART_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 5, 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.*

### 25.4.2 Data Structure Documentation

#### 25.4.2.1 struct `lpuart_rtos_config_t`

##### Data Fields

- `LPUART_Type * base`  
*UART base address.*
- `uint32_t srclk`  
*UART source clock in Hz.*
- `uint32_t baudrate`  
*Desired communication speed.*
- `lpuart_parity_mode_t parity`  
*Parity setting.*

- `lpuart_stop_bit_count_t stopbits`  
*Number of stop bits to use.*
- `uint8_t * buffer`  
*Buffer for background reception.*
- `uint32_t buffer_size`  
*Size of buffer for background reception.*
- `bool enableRxRTS`  
*RX RTS enable.*
- `bool enableTxCTS`  
*TX CTS enable.*
- `lpuart_transmit_cts_source_t txCtsSource`  
*TX CTS source.*
- `lpuart_transmit_cts_config_t txCtsConfig`  
*TX CTS configure.*

### 25.4.3 Macro Definition Documentation

**25.4.3.1 `#define FSL_LPUART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))`**

### 25.4.4 Function Documentation

**25.4.4.1 `int LPUART_RTOS_Init ( lpuart_rtos_handle_t * handle, lpuart_handle_t * t_handle, const lpuart_rtos_config_t * cfg )`**

Parameters

|                       |                                                                                      |
|-----------------------|--------------------------------------------------------------------------------------|
| <code>handle</code>   | The RTOS LPUART handle, the pointer to an allocated space for RTOS context.          |
| <code>t_handle</code> | The pointer to an allocated space to store the transactional layer internal state.   |
| <code>cfg</code>      | The pointer to the parameters required to configure the LPUART after initialization. |

Returns

0 succeed, others failed

**25.4.4.2 `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. |
|---------------|-------------------------|

#### 25.4.4.3 int LPUART\_RTOS\_Send ( Ipuart\_rtos\_handle\_t \* *handle*, uint8\_t \* *buffer*, uint32\_t *length* )

This function sends data. It is an synchronous API. If the hardware buffer is full, the task is in the blocked state.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>handle</i> | The RTOS LPUART handle.        |
| <i>buffer</i> | The pointer to buffer to send. |
| <i>length</i> | The number of bytes to send.   |

#### 25.4.4.4 int LPUART\_RTOS\_Receive ( Ipuart\_rtos\_handle\_t \* *handle*, uint8\_t \* *buffer*, uint32\_t *length*, size\_t \* *received* )

This function receives data from LPUART. It is an synchronous API. If any data is immediately available it is returned immediately and the number of bytes received.

Parameters

|                 |                                                                                  |
|-----------------|----------------------------------------------------------------------------------|
| <i>handle</i>   | The RTOS LPUART handle.                                                          |
| <i>buffer</i>   | The pointer to buffer where to write received data.                              |
| <i>length</i>   | The number of bytes to receive.                                                  |
| <i>received</i> | The pointer to a variable of size_t where the number of received data is filled. |

## 25.5 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.

### 25.5.1 Function groups

#### 25.5.1.1 LPUART CMSIS GetVersion Operation

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

#### 25.5.1.2 LPUART CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

#### 25.5.1.3 LPUART CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialized the lpuart instance . And this API must be called before you configure a lpuart instance or after you Deinit a lpuart instance.The right steps to start an instance is that you must initialize the instance which been selected firstly,then you can power on the instance.After these all have been done,you can configure the instance by using control operation.If you want to Uninitialize the instance, you must power off the instance first.

#### 25.5.1.4 LPUART CMSIS Transfer Operation

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

#### 25.5.1.5 LPUART CMSIS Status Operation

This function group gets the LPUART transfer status.

### **25.5.1.6 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 26

## OCOTP: On Chip One-Time Programmable controller.

### 26.1 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.

### 26.2 OCOTP function group

The OCOTP driver support operaing API to allow read and write the fuse map.

#### 26.2.1 Initialization and de-initialization

The funciton [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.

#### 26.2.2 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.

### 26.3 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

### Data Structures

- struct [ocotp\\_timing\\_t](#)  
*OCOTP timing structure.* [More...](#)

## 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.*

## 26.4 Data Structure Documentation

### 26.4.1 struct ocotp\_timing\_t

Note that, these value are used for calcalating the read/write timings. And the values should statisfy below rules:

`Tsp_rd=(WAIT+1)/ipg_clk_freq` should be  $\geq 150\text{ns}$ ; `Tsp_pgm=(RELAX+1)/ipg_clk_freq` should be  $\geq 100\text{ns}$ ; `Trd = ((STROBE_READ+1)- 2*(RELAX_READ+1)) /ipg_clk_freq`, The `Trd` is required to be larger than 40 ns. `Tpgm = ((STROBE_PROG+1)- 2*(RELAX_PROG+1)) /ipg_clk_freq`; The `Tpgm` should be configured within the range of 9000 ns < `Tpgm` < 11000 ns;

## Data Fields

- `uint32_t wait`  
*Wait time value to fill in the TIMING register.*
- `uint32_t relax`  
*Relax time value to fill in the TIMING register.*
- `uint32_t strobe_prog`  
*Strobe program time value to fill in the TIMING register.*
- `uint32_t strobe_read`  
*Strobe read time value to fill in the TIMING register.*

### Field Documentation

- (1) `uint32_t ocotp_timing_t::wait`
- (2) `uint32_t ocotp_timing_t::relax`
- (3) `uint32_t ocotp_timing_t::strobe_prog`
- (4) `uint32_t ocotp_timing_t::strobe_read`

## 26.5 Macro Definition Documentation

`#define FSL_OCOTP_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))`

## 26.6 Enumeration Type Documentation

### 26.6.1 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.

## 26.7 Function Documentation

`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. |

### 26.7.2 void OCOTP\_Deinit( OCOTP\_Type \* *base* )

Return values

|                        |                                                    |
|------------------------|----------------------------------------------------|
| <i>kStatus_Success</i> | upon successful execution, error status otherwise. |
|------------------------|----------------------------------------------------|

### 26.7.3 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. |
|-------------|------------------------------------|

### 26.7.4 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. |
|-------------|------------------------------------|

### 26.7.5 static void OCOTP\_ClearErrorStatus ( OCOTP\_Type \* *base* ) [inline], [static]

Parameters

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

### 26.7.6 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 addess. |
|-------------|-------------------------------|

Return values

|                                   |                 |
|-----------------------------------|-----------------|
| <i>kStatus_Success</i>            | Reload success. |
| <i>kStatus_OCOTP_Reload_Error</i> | Reload failed.  |

### 26.7.7 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.

### 26.7.8 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. |

### 26.7.9 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. |
|--------------|------------------------------------------------------------------|

### 26.7.10 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.                                                   |

### 26.7.11 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 27

## OTFAD: On The Fly AES-128 Decryption Driver

### 27.1 Overview

The MCUXpresso SDK provides a peripheral driver for On The Fly AES-128 Decryption (OTFAD) module of MCUXpresso SDK devices.

This example code shows how to decrypt ciphertext in external memory using the OTFAD driver.

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

### Data Structures

- struct [otfad\\_encryption\\_config\\_t](#)  
*OTFAD encryption configuration structure. [More...](#)*
- struct [otfad\\_config\\_t](#)  
*OTFAD configuration structure. [More...](#)*

### Enumerations

- enum {  
  [kStatus\\_OTFAD\\_ResRegAccessMode](#) = MAKE\_STATUS(kStatusGroup\_OTFAD, 0),  
  [kStatus\\_OTFAD\\_AddressError](#) = MAKE\_STATUS(kStatusGroup\_OTFAD, 1),  
  [kStatus\\_OTFAD\\_RegionOverlap](#),  
  [kStatus\\_OTFAD\\_RegionMiss](#) }  
*Status codes for the OTFAD driver.*
- enum {  
  [kOTFAD\\_Context\\_0](#) = 0U,  
  [kOTFAD\\_Context\\_1](#) = 1U,  
  [kOTFAD\\_Context\\_2](#) = 2U,  
  [kOTFAD\\_Context\\_3](#) = 3U }  
*OTFAD context type.*
- enum {  
  [kOTFAD\\_NRM](#) = 0x00U,  
  [kOTFAD\\_SVM](#) = 0x02U,  
  [kOTFAD\\_LDM](#) = 0x03U }  
*OTFAD operate mode.*

### Driver version

- #define [FSL\\_OTFAD\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2U, 1U, 3U))  
*Driver version.*

## Initialization and deinitialization

- void **OTFAD\_GetDefaultConfig** (otfad\_config\_t \*config)  
*OTFAD module initialization function.*
- void **OTFAD\_Init** (OTFAD\_Type \*base, const otfad\_config\_t \*config)  
*OTFAD module initialization function.*
- void **OTFAD\_Deinit** (OTFAD\_Type \*base)  
*Deinitializes the OTFAD.*

## Status

- static uint32\_t **OTFAD\_GetOperateMode** (OTFAD\_Type \*base)  
*OTFAD module get operate mode.*
- static uint32\_t **OTFAD\_GetStatus** (OTFAD\_Type \*base)  
*OTFAD module get status.*

## functional

- status\_t **OTFAD\_SetEncryptionConfig** (OTFAD\_Type \*base, const otfad\_encryption\_config\_t \*config)  
*OTFAD module set encryption configuration.*
- status\_t **OTFAD\_GetEncryptionConfig** (OTFAD\_Type \*base, otfad\_encryption\_config\_t \*config)  
*OTFAD module get encryption configuration.*
- status\_t **OTFAD\_HitDetermination** (OTFAD\_Type \*base, uint32\_t address, uint8\_t \*contextIndex)  
*OTFAD module do hit determination.*

## 27.2 Data Structure Documentation

### 27.2.1 struct otfad\_encryption\_config\_t

#### Data Fields

- bool **valid**  
*The context is valid or not.*
- bool **AESdecryption**  
*AES decryption enable.*
- uint8\_t **readOnly**  
*read write attribute for the entire set of context registers*
- uint8\_t **contextIndex**  
*OTFAD context index.*
- uint32\_t **startAddr**  
*Start address.*
- uint32\_t **endAddr**  
*End address.*
- uint32\_t **key** [4]  
*Encryption key.*
- uint32\_t **counter** [2]  
*Encryption counter.*

## 27.2.2 struct otfad\_config\_t

### Data Fields

- bool `forceError`  
*Forces the OTFAD's key blob error flag (SR[KBERR]) to be asserted.*
- bool `forceSVM`  
*Force entry into SVM after a write.*
- bool `forceLDM`  
*Force entry into LDM after a write.*
- bool `keyBlobScramble`  
*Key blob KEK scrambling.*
- bool `keyBlobProcess`  
*Key blob processing.*
- bool `startKeyBlobProcessing`  
*key blob processing is initiated*
- bool `restrictedRegAccess`  
*Restricted register access enable.*
- bool `enableOTFAD`  
*OTFAD has decryption enabled.*

## 27.3 Macro Definition Documentation

### 27.3.1 #define FSL\_OTFAD\_DRIVER\_VERSION (MAKE\_VERSION(2U, 1U, 3U))

## 27.4 Enumeration Type Documentation

### 27.4.1 anonymous enum

Enumerator

`kStatus_OTFAD_ResRegAccessMode` Restricted register mode.

`kStatus_OTFAD_AddressError` End address less than start address.

`kStatus_OTFAD_RegionOverlap` the OTFAD does not support any form of memory region overlap, for system accesses that hit in multiple contexts or no contexts, the fetched data is simply bypassed

`kStatus_OTFAD_RegionMiss` For accesses that hit in a single context, but not the selected one.

### 27.4.2 anonymous enum

Enumerator

`kOTFAD_Context_0` context 0

`kOTFAD_Context_1` context 1

`kOTFAD_Context_2` context 2

`kOTFAD_Context_3` context 3

### 27.4.3 anonymous enum

Enumerator

- kOTFAD\_NRM*** Normal Mode.
- kOTFAD\_SVM*** Security Violation Mode.
- kOTFAD\_LDM*** Logically Disabled Mode.

## 27.5 Function Documentation

### 27.5.1 void OTFAD\_GetDefaultConfig ( *otfad\_config\_t* \* *config* )

Parameters

|               |                      |
|---------------|----------------------|
| <i>config</i> | OTFAD configuration. |
|---------------|----------------------|

### 27.5.2 void OTFAD\_Init ( *OTFAD\_Type* \* *base*, *const otfad\_config\_t* \* *config* )

Parameters

|               |                      |
|---------------|----------------------|
| <i>base</i>   | OTFAD base address.  |
| <i>config</i> | OTFAD configuration. |

### 27.5.3 static uint32\_t OTFAD\_GetOperateMode ( *OTFAD\_Type* \* *base* ) [inline], [static]

Parameters

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

### 27.5.4 static uint32\_t OTFAD\_GetStatus ( *OTFAD\_Type* \* *base* ) [inline], [static]

Parameters

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

### 27.5.5 status\_t OTFAD\_SetEncryptionConfig ( OTFAD\_Type \* *base*, const otfad\_encryption\_config\_t \* *config* )

Parameters

|               |                           |
|---------------|---------------------------|
| <i>base</i>   | OTFAD base address.       |
| <i>config</i> | encryption configuration. |

Note: if enable keyblob process, the first 256 bytes external memory is use for keyblob data, so this region shouldn't be in OTFAD region.

### 27.5.6 status\_t OTFAD\_GetEncryptionConfig ( OTFAD\_Type \* *base*, otfad\_encryption\_config\_t \* *config* )

Parameters

|               |                           |
|---------------|---------------------------|
| <i>base</i>   | OTFAD base address.       |
| <i>config</i> | encryption configuration. |

Note: if enable keyblob process, the first 256 bytes external memory is use for keyblob data, so this region shouldn't be in OTFAD region.

### 27.5.7 status\_t OTFAD\_HitDetermination ( OTFAD\_Type \* *base*, uint32\_t *address*, uint8\_t \* *contextIndex* )

Parameters

|                     |                                                                     |
|---------------------|---------------------------------------------------------------------|
| <i>base</i>         | OTFAD base address.                                                 |
| <i>address</i>      | the physical address space assigned to the QuadSPI(FlexSPI) module. |
| <i>contextIndex</i> | hitted context region index.                                        |

Returns

status, such as kStatus\_Success or kStatus\_OTFAD\_ResRegAccessMode.

# Chapter 28

## PIT: Periodic Interrupt Timer

### 28.1 Overview

The MCUXpresso SDK provides a driver for the Periodic Interrupt Timer (PIT) of MCUXpresso SDK devices.

### 28.2 Function groups

The PIT driver supports operating the module as a time counter.

#### 28.2.1 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.

#### 28.2.2 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.

#### 28.2.3 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.

## 28.2.4 Status

Provides functions to get and clear the PIT status.

## 28.2.5 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

## 28.3 Typical use case

### 28.3.1 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.*

# Functions

- `uint64_t PIT_GetLifetimeTimerCount (PIT_Type *base)`  
*Reads the current lifetime counter value.*

## Driver version

- `#define FSL_PIT_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))`  
*PIT Driver Version 2.0.4.*

## Initialization and deinitialization

- void PIT\_Init(PIT\_Type \*base, const pit\_config\_t \*config)

- Ungates the PIT clock, enables the PIT module, and configures the peripheral for basic operations.
- void [PIT\\_Deinit](#) (PIT\_Type \*base)  
*Gates the PIT clock and disables the PIT module.*
- static void [PIT\\_GetDefaultConfig](#) (pit\_config\_t \*config)  
*Fills in the PIT configuration structure with the default settings.*
- static void [PIT\\_SetTimerChainMode](#) (PIT\_Type \*base, pit\_chnl\_t channel, bool enable)  
*Enables or disables chaining a timer with the previous timer.*

## Interrupt Interface

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

## Status Interface

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

## Read and Write the timer period

- static void [PIT\\_SetTimerPeriod](#) (PIT\_Type \*base, pit\_chnl\_t channel, uint32\_t count)  
*Sets the timer period in units of count.*
- static uint32\_t [PIT\\_GetCurrentTimerCount](#) (PIT\_Type \*base, pit\_chnl\_t channel)  
*Reads the current timer counting value.*

## Timer Start and Stop

- static void [PIT\\_StartTimer](#) (PIT\_Type \*base, pit\_chnl\_t channel)  
*Starts the timer counting.*
- static void [PIT\\_StopTimer](#) (PIT\_Type \*base, pit\_chnl\_t channel)  
*Stops the timer counting.*

## 28.4 Data Structure Documentation

### 28.4.1 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*

## 28.5 Enumeration Type Documentation

### 28.5.1 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.

### 28.5.2 enum pit\_interrupt\_enable\_t

Enumerator

- kPIT\_TimerInterruptEnable*** Timer interrupt enable.

### 28.5.3 enum pit\_status\_flags\_t

Enumerator

- kPIT\_TimerFlag*** Timer flag.

## 28.6 Function Documentation

### 28.6.1 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 |

## 28.6.2 void PIT\_Deinit ( PIT\_Type \* *base* )

Parameters

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

## 28.6.3 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. |
|---------------|-----------------------------------------|

## 28.6.4 static void PIT\_SetTimerChainMode ( PIT\_Type \* *base*, pit\_chnl\_t *channel*, bool *enable* ) [inline], [static]

When a timer has a chain mode enabled, it only counts after the previous timer has expired. If the timer n-1 has counted down to 0, counter n decrements the value by one. Each timer is 32-bits, which allows the developers to chain timers together and form a longer timer (64-bits and larger). The first timer (timer 0) can't be chained to any other timer.

Parameters

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

|                |                                                                                                                                |
|----------------|--------------------------------------------------------------------------------------------------------------------------------|
| <i>channel</i> | Timer channel number which is chained with the previous timer                                                                  |
| <i>enable</i>  | Enable or disable chain. true: Current timer is chained with the previous timer. false: Timer doesn't chain with other timers. |

### 28.6.5 static void PIT\_EnableInterrupts ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel*, **uint32\_t** *mask* ) [inline], [static]

Parameters

|                |                                                                                                                     |
|----------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | PIT peripheral base address                                                                                         |
| <i>channel</i> | Timer channel number                                                                                                |
| <i>mask</i>    | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">pit_interrupt_enable_t</a> |

### 28.6.6 static void PIT\_DisableInterrupts ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel*, **uint32\_t** *mask* ) [inline], [static]

Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | PIT peripheral base address                                                                                          |
| <i>channel</i> | Timer channel number                                                                                                 |
| <i>mask</i>    | The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">pit_interrupt_enable_t</a> |

### 28.6.7 static **uint32\_t** PIT\_GetEnabledInterrupts ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel* ) [inline], [static]

Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number        |

Returns

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

28.6.8 **static uint32\_t PIT\_GetStatusFlags ( PIT\_Type \* *base*, pit\_chnl\_t *channel* )**  
[**inline**], [**static**]

Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number        |

Returns

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

### 28.6.9 static void PIT\_ClearStatusFlags ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel*, **uint32\_t** *mask* ) [inline], [static]

Parameters

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

### 28.6.10 static void PIT\_SetTimerPeriod ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel*, **uint32\_t** *count* ) [inline], [static]

Timers begin counting from the value set by this function until it reaches 0, then it generates an interrupt and load this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

Note

Users can call the utility macros provided in `fsl_common.h` to convert to ticks.

Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number        |

|              |                                |
|--------------|--------------------------------|
| <i>count</i> | Timer period in units of ticks |
|--------------|--------------------------------|

### 28.6.11 static uint32\_t PIT\_GetCurrentTimerCount ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel* ) [inline], [static]

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note

Users can call the utility macros provided in fsl\_common.h to convert ticks to usec or msec.

Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number        |

Returns

Current timer counting value in ticks

### 28.6.12 static void PIT\_StartTimer ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel* ) [inline], [static]

After calling this function, timers load period value, count down to 0 and then load the respective start value again. Each time a timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number.       |

### 28.6.13 static void PIT\_StopTimer ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel* ) [inline], [static]

This function stops every timer counting. Timers reload their periods respectively after the next time they call the PIT\_DRV\_StartTimer.

Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number.       |

### 28.6.14 `uint64_t PIT_GetLifetimeTimerCount ( PIT_Type * base )`

The lifetime timer is a 64-bit timer which chains timer 0 and timer 1 together. Timer 0 and 1 are chained by calling the `PIT_SetTimerChainMode` before using this timer. The period of lifetime timer is equal to the "period of timer 0 \* period of timer 1". For the 64-bit value, the higher 32-bit has the value of timer 1, and the lower 32-bit has the value of timer 0.

Parameters

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

Returns

Current lifetime timer value

# Chapter 29

## PMU: Power Management Unit

### 29.1 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)

- static void [PMU\\_2P5SetBrownoutOffsetVoltage](#) (PMU\_Type \*base, uint32\_t value)
 

*Adjust the 1P1 regulator output voltage.*
- static void [PMU\\_2P5EnablePullDown](#) (PMU\_Type \*base, bool enable)
 

*Adjust the 2P5 regulator brownout offset voltage.*
- 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 pull-down circuitry in the 2P5 regulator.*
- static void [PMU\\_2P5nableBrownout](#) (PMU\_Type \*base, bool enable)
 

*Enable the current-limit circuitry in the 2P5 regulator.*
- static void [PMU\\_2P5EnableOutput](#) (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.*

## 29.2 Macro Definition Documentation

### 29.2.1 #define FSL\_PMU\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))

Version 2.1.1.

## 29.3 Enumeration Type Documentation

### 29.3.1 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.

### 29.3.2 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.

### 29.3.3 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.

### 29.3.4 enum pmu\_core\_reg\_voltage\_ramp\_rate\_t

Enumerator

***kPMU\_CoreRegVoltageRampRateFast*** Fast.

***kPMU\_CoreRegVoltageRampRateMediumFast*** Medium Fast.

***kPMU\_CoreRegVoltageRampRateMediumSlow*** Medium Slow.

***kPMU\_CoreRegVoltageRampRateSlow*** Slow.

### 29.3.5 enum pmu\_power\_bandgap\_t

Enumerator

***kPMU\_NormalPowerBandgap*** Normal power bandgap.

***kPMU\_LowPowerBandgap*** Low power bandgap.

## 29.4 Function Documentation

### 29.4.1 uint32\_t PMU\_GetStatusFlags ( PMU\_Type \* base )

Parameters

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

Returns

PMU status flags. It indicate if regulator output of 1P1,3P0 and 2P5 is ok and brownout output of 1P1,3P0 and 2P5 is detected.

#### 29.4.2 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> . |

#### 29.4.3 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.   |

#### 29.4.4 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. |

#### 29.4.5 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. |

#### 29.4.6 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.   |

#### 29.4.7 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.   |

#### 29.4.8 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.   |

#### 29.4.9 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.   |

#### 29.4.10 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. |

#### 29.4.11 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. |

#### 29.4.12 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. |

#### 29.4.13 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.   |

#### 29.4.14 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.   |

#### 29.4.15 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.   |

#### 29.4.16 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.   |

#### 29.4.17 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. |

#### 29.4.18 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. |

**29.4.19 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.   |

**29.4.20 static void PMU\_2P1EnablePullDown ( PMU\_Type \* *base*, bool *enable* )  
[inline], [static]**

**Deprecated** Do not use this function. It has been superceded by [PMU\\_2P5EnablePullDown](#).

**29.4.21 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.   |

**29.4.22 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.   |

**29.4.23 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.   |

**29.4.24 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.   |

**29.4.25 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> . |

### 29.4.26 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 |

### 29.4.27 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 30

## PWM: Pulse Width Modulator

### 30.1 Overview

The MCUXpresso SDK provides a driver for the Pulse Width Modulator (PWM) of MCUXpresso SDK devices.

### 30.2 PWM: Pulse Width Modulator

#### 30.2.1 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.

#### 30.2.2 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.

#### 30.2.3 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.

#### 30.2.4 Fault operation

The function [PWM\\_SetupFault\(\)](#) sets up the properties for each fault.

### 30.2.5 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.

### 30.2.6 Status

Provide functions to get and clear the PWM status.

### 30.2.7 Interrupt

Provide functions to enable/disable PWM interrupts and get current enabled interrupts.

## 30.3 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/pwmThe 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

## 30.4 Typical use case

### 30.4.1 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.*

## 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_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_DMAResetDisable` = 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_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 valueRegister)  
*Enables the PWM output trigger.*
- static void `PWM_DeactivateOutputTrigger` (PWM\_Type \*base, `pwm_submodule_t` subModule, uint16\_t valueRegister)  
*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 LDOKE 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.*

## Driver version

- #define `FSL_PWM_DRIVER_VERSION` (MAKE\_VERSION(2, 2, 1))  
*Version 2.2.1.*

## 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.*
- 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\_EnableDMACapture** (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\_EnableDMAWrite** (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.*

## 30.5 Data Structure Documentation

### 30.5.1 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.*

#### Field Documentation

##### (1) uint8\_t pwm\_signal\_param\_t::dutyCyclePercent

100=always active signal (100% duty cycle)

### 30.5.2 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](#)

- **pwm\_clock\_prescale\_t prescale**  
*Clock source for the counter.*
- **pwm\_chnl\_pair\_operation\_t pairOperation**  
*Pre-scaler to divide down the clock.*
- **pwm\_register\_reload\_t reloadLogic**  
*Channel pair in independent or complementary mode.*
- **pwm\_reload\_source\_select\_t reloadSelect**  
*PWM Reload logic setup.*
- **pwm\_reload\_frequency\_t reloadFrequency**  
*Reload source select.*
- **pwm\_force\_output\_trigger\_t forceTrigger**  
*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.*

### 30.5.3 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.*

### 30.5.4 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.*

### 30.5.5 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

**30.6 Enumeration Type Documentation****30.6.1 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.

**30.6.2 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.

**30.6.3 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.

### 30.6.4 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.

### 30.6.5 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.

### 30.6.6 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 submodule0 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.

### 30.6.7 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.

### 30.6.8 enum pwm\_load\_frequency\_t

Enumerator

***kPWM\_LoadEveryOportunity*** Every PWM opportunity.

***kPWM\_LoadEvery2Oportunity*** Every 2 PWM opportunities.

***kPWM\_LoadEvery3Oportunity*** Every 3 PWM opportunities.

***kPWM\_LoadEvery4Oportunity*** Every 4 PWM opportunities.

***kPWM\_LoadEvery5Oportunity*** Every 5 PWM opportunities.

***kPWM\_LoadEvery6Oportunity*** Every 6 PWM opportunities.

***kPWM\_LoadEvery7Oportunity*** Every 7 PWM opportunities.

***kPWM\_LoadEvery8Oportunity*** Every 8 PWM opportunities.

***kPWM\_LoadEvery9Oportunity*** Every 9 PWM opportunities.

***kPWM\_LoadEvery10Oportunity*** Every 10 PWM opportunities.

***kPWM\_LoadEvery11Oportunity*** Every 11 PWM opportunities.

***kPWM\_LoadEvery12Oportunity*** Every 12 PWM opportunities.

***kPWM\_LoadEvery13Oportunity*** Every 13 PWM opportunities.

***kPWM\_LoadEvery14Oportunity*** Every 14 PWM opportunities.

***kPWM\_LoadEvery15Oportunity*** Every 15 PWM opportunities.

***kPWM\_LoadEvery16Oportunity*** Every 16 PWM opportunities.

### 30.6.9 enum pwm\_fault\_input\_t

Enumerator

***kPWM\_Fault\_0*** Fault 0 input pin.

***kPWM\_Fault\_1*** Fault 1 input pin.

***kPWM\_Fault\_2*** Fault 2 input pin.

***kPWM\_Fault\_3*** Fault 3 input pin.

**30.6.10 enum pwm\_fault\_disable\_t**

Enumerator

- kPWM\_FaultDisable\_0*** Fault 0 disable mapping.
- kPWM\_FaultDisable\_1*** Fault 1 disable mapping.
- kPWM\_FaultDisable\_2*** Fault 2 disable mapping.
- kPWM\_FaultDisable\_3*** Fault 3 disable mapping.

**30.6.11 enum pwm\_input\_capture\_edge\_t**

Enumerator

- kPWM\_Disable*** Disabled.
- kPWM\_FallingEdge*** Capture on falling edge only.
- kPWM\_RisingEdge*** Capture on rising edge only.
- kPWM\_RiseAndFallEdge*** Capture on rising or falling edge.

**30.6.12 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\_EXTA signal is used by the deadtime logic.

**30.6.13 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.

**30.6.14 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.

**30.6.15 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.

**30.6.16 enum pwm\_interrupt\_enable\_t**

Enumerator

- kPWM\_CompareVal0InterruptEnable*** PWM VAL0 compare interrupt.
- kPWM\_CompareVal1InterruptEnable*** PWM VAL1 compare interrupt.
- kPWM\_CompareVal2InterruptEnable*** PWM VAL2 compare interrupt.
- kPWM\_CompareVal3InterruptEnable*** PWM VAL3 compare interrupt.
- kPWM\_CompareVal4InterruptEnable*** PWM VAL4 compare interrupt.
- kPWM\_CompareVal5InterruptEnable*** PWM VAL5 compare interrupt.
- kPWM\_CaptureX0InterruptEnable*** PWM capture X0 interrupt.
- kPWM\_CaptureX1InterruptEnable*** PWM capture X1 interrupt.
- kPWM\_CaptureB0InterruptEnable*** PWM capture B0 interrupt.
- kPWM\_CaptureB1InterruptEnable*** PWM capture B1 interrupt.
- kPWM\_CaptureA0InterruptEnable*** PWM capture A0 interrupt.
- kPWM\_CaptureA1InterruptEnable*** PWM capture A1 interrupt.
- kPWM\_ReloadInterruptEnable*** PWM reload interrupt.
- kPWM\_ReloadErrorInterruptEnable*** PWM reload error interrupt.
- kPWM\_Fault0InterruptEnable*** PWM fault 0 interrupt.
- kPWM\_Fault1InterruptEnable*** PWM fault 1 interrupt.
- kPWM\_Fault2InterruptEnable*** PWM fault 2 interrupt.
- kPWM\_Fault3InterruptEnable*** PWM fault 3 interrupt.

### 30.6.17 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.           |
| <i>kPWM_Fault2Flag</i>      | PWM fault 2 flag.           |
| <i>kPWM_Fault3Flag</i>      | PWM fault 3 flag.           |

### 30.6.18 enum pwm\_dma\_enable\_t

Enumerator

|                                |                     |
|--------------------------------|---------------------|
| <i>kPWM_CaptureX0DMAEnable</i> | PWM capture X0 DMA. |
| <i>kPWM_CaptureX1DMAEnable</i> | PWM capture X1 DMA. |
| <i>kPWM_CaptureB0DMAEnable</i> | PWM capture B0 DMA. |
| <i>kPWM_CaptureB1DMAEnable</i> | PWM capture B1 DMA. |
| <i>kPWM_CaptureA0DMAEnable</i> | PWM capture A0 DMA. |
| <i>kPWM_CaptureA1DMAEnable</i> | PWM capture A1 DMA. |

### 30.6.19 enum pwm\_dma\_source\_select\_t

Enumerator

|                                 |                                                                |
|---------------------------------|----------------------------------------------------------------|
| <i>kPWM_DMAResetDisable</i>     | Read DMA requests disabled.                                    |
| <i>kPWM_DMAWatermarksEnable</i> | Exceeding a FIFO watermark sets the DMA read request.          |
| <i>kPWM_DMALocalSync</i>        | A local sync (VAL1 matches counter) sets the read DMA request. |
| <i>kPWM_DMALocalReload</i>      | A local reload (STS[RF] being set) sets the read DMA request.  |

### 30.6.20 enum pwm\_watermark\_control\_t

Enumerator

***kPWM\_FIFOWatermarksOR*** Selected FIFO watermarks are OR'ed together.

***kPWM\_FIFOWatermarksAND*** Selected FIFO watermarks are AND'ed together.

### 30.6.21 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.

### 30.6.22 enum pwm\_level\_select\_t

Enumerator

***kPWM\_HighTrue*** High level represents "on" or "active" state.

***kPWM\_LowTrue*** Low level represents "on" or "active" state.

### 30.6.23 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.

### 30.6.24 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.

### 30.6.25 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.

### 30.6.26 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.

## 30.7 Function Documentation

### 30.7.1 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.

### 30.7.2 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 |

**30.7.3 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_LoadEveryOportunity;
* config->reloadLogic = kPWM_ReloadImmediate;
* config->pairOperation = kPWM_Independent;
*
```

## Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>config</i> | Pointer to user's PWM config structure. |
|---------------|-----------------------------------------|

**30.7.4 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)                                                                                                      |
| <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 kStatusFail if there was error setting up the signal; kStatusSuccess otherwise

### 30.7.5 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 dutycycle 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) |

### 30.7.6 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 dutycycle 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) |

### 30.7.7 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>inputCaptureParams</i> | Parameters passed in to set up the input pin |

### 30.7.8 void PWM\_SetupFaultInputFilter ( PWM\_Type \* *base*, const pwm\_fault\_input\_filter\_param\_t \* *faultInputFilterParams* )

Parameters

|                               |                                                        |
|-------------------------------|--------------------------------------------------------|
| <i>base</i>                   | PWM peripheral base address                            |
| <i>faultInputFilterParams</i> | Parameters passed in to set up the fault input filter. |

### 30.7.9 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 |

### 30.7.10 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. |
|---------------|-----------------------------------------------|

### 30.7.11 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 |

### 30.7.12 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> |

### 30.7.13 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> |

### 30.7.14 **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](#)

### 30.7.15 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_watermark_control</i> | PWM FIFO watermark and control |

**30.7.16 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_source_select</i> | PWM capture DMA enable source select |

**30.7.17 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                                                    |

**30.7.18 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 |

### 30.7.19 **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](#)

### 30.7.20 **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> |

### 30.7.21 **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>subModulesToStart</i> | PWM submodules to start. This is a logical OR of members of the enumeration <a href="#">pwm_module_control_t</a> |

### 30.7.22 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>subModulesToStop</i> | PWM submodules to stop. This is a logical OR of members of the enumeration <a href="#">pwm_module_control_t</a> |

### 30.7.23 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 |

### 30.7.24 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>valueRegister-Mask</i> | Value register mask that will activate one or more (VAL0-5) trigger enumeration <a href="#">_pwm_value_register_mask</a> |

**30.7.25 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>valueRegister-Mask</i> | Value register mask that will Deactivate one or more (VAL0-5) trigger enumeration <a href="#">_pwm_value_register_mask</a> |

**30.7.26 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. |

### 30.7.27 static void PWM\_SetPwmLdok ( **PWM\_Type** \* *base*, **uint8\_t** *subModulesToUpdate*, **bool** *value* ) [inline], [static]

Set LDOOK 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 LDOOK bit for the submodule list; false: Clear LDOOK bit                                                                     |

### 30.7.28 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     |

### 30.7.29 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], [i 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> |

# Chapter 31

## RTWDOG: 32-bit Watchdog Timer

### 31.1 Overview

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

### 31.2 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 clockFreqInHz)  
*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)

- static void [RTWDOG\\_SetWindowValue](#) (RTWDOG\_Type \*base, uint16\_t windowValue)
 

*Sets the RTWDOG timeout 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.*

## 31.3 Data Structure Documentation

### 31.3.1 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.*

### 31.3.2 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.*

## 31.4 Macro Definition Documentation

**31.4.1 #define FSL\_RTWDOG\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 2))**

## 31.5 Enumeration Type Documentation

### 31.5.1 enum rtwdog\_clock\_source\_t

Enumerator

*kRTWDOG\_ClockSource0* Clock source 0.

*kRTWDOG\_ClockSource1* Clock source 1.

*kRTWDOG\_ClockSource2* Clock source 2.

*kRTWDOG\_ClockSource3* Clock source 3.

### 31.5.2 enum rtwdog\_clock\_prescaler\_t

Enumerator

*kRTWDOG\_ClockPrescalerDivide1* Divided by 1.

*kRTWDOG\_ClockPrescalerDivide256* Divided by 256.

### 31.5.3 enum rtwdog\_test\_mode\_t

Enumerator

*kRTWDOG\_TestModeDisabled* Test Mode disabled.

*kRTWDOG\_UserModeEnabled* User Mode enabled.

*kRTWDOG\_LowByteTest* Test Mode enabled, only low byte is used.

*kRTWDOG\_HighByteTest* Test Mode enabled, only high byte is used.

### 31.5.4 enum \_rtwdog\_interrupt\_enable\_t

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

Enumerator

*kRTWDOG\_InterruptEnable* Interrupt is generated before forcing a reset.

### 31.5.5 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.

## 31.6 Function Documentation

### 31.6.1 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](#)

### 31.6.2 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. |

### 31.6.3 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. |
|-------------|---------------------------------|

### 31.6.4 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. |
|-------------|---------------------------------|

### 31.6.5 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 |
|-------------|--------------------------------|

### 31.6.6 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> |

### 31.6.7 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> |

### 31.6.8 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.

### 31.6.9 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. |

### 31.6.10 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. |

### 31.6.11 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> |

**31.6.12 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. |

**31.6.13 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.            |

**31.6.14 \_\_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 |
|-------------|--------------------------------|

### **31.6.15 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 |
|-------------|--------------------------------|

### **31.6.16 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 32

## SAI: Serial Audio Interface

### 32.1 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.

### 32.2 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`

### 32.3 Typical use case

#### 32.3.1 SAI Send/receive using an interrupt method

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

#### 32.3.2 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](#)

## 32.4 SAI Driver

### 32.4.1 Overview

#### Data Structures

- struct [sai\\_config\\_t](#)  
*SAI user configuration structure. [More...](#)*
- struct [sai\\_transfer\\_format\\_t](#)  
*sai transfer format [More...](#)*
- struct [sai\\_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_FifoCombineModeEnabledReadWrite }
- 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, 4)`)
- Version 2.3.4.*

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_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.

- `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` type)
`Do software reset or FIFO reset .`
- void `SAI_RxSoftwareReset` (I2S_Type *base, `sai_reset_type_t` type)
`Do software reset or FIFO reset .`
- void `SAI_TxSetChannelFIFOMask` (I2S_Type *base, `uint8_t` mask)
`Set the Tx channel FIFO enable mask.`
- void `SAI_RxSetChannelFIFOMask` (I2S_Type *base, `uint8_t` mask)
`Set the Rx channel FIFO enable mask.`
- void `SAI_TxSetDataOrder` (I2S_Type *base, `sai_data_order_t` order)
`Set the Tx data order.`
- void `SAI_RxSetDataOrder` (I2S_Type *base, `sai_data_order_t` order)
`Set the Rx data order.`
- void `SAI_TxSetBitClockPolarity` (I2S_Type *base, `sai_clock_polarity_t` polarity)

- Set the Tx data order.
- void [SAI_RxSetBitClockPolarity](#) (I2S_Type *base, sai_clock_polarity_t polarity)
 - Set the Rx data order.
- void [SAI_TxSetFrameSyncPolarity](#) (I2S_Type *base, sai_clock_polarity_t polarity)
 - Set the Tx data order.
- void [SAI_RxSetFrameSyncPolarity](#) (I2S_Type *base, sai_clock_polarity_t polarity)
 - Set the Rx data order.
- 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 uint32_t [SAI_TxGetDataRegisterAddress](#) (I2S_Type *base, uint32_t channel)
 - Gets the SAI Tx data register address.
- static uint32_t [SAI_RxGetDataRegisterAddress](#) (I2S_Type *base, uint32_t channel)
 - Gets the SAI Rx data register address.

Bus Operations

- void [SAI_TxSetFormat](#) (I2S_Type *base, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
 - Configures the SAI Tx audio format.
- void [SAI_RxSetFormat](#) (I2S_Type *base, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
 - Configures the SAI Rx audio format.

- void **SAI_WriteBlocking** (I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data using a blocking method.
- void **SAI_WriteMultiChannelBlocking** (I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data to multi channel using a blocking method.
- static void **SAI_WriteData** (I2S_Type *base, uint32_t channel, uint32_t data)

Writes data into SAI FIFO.
- void **SAI_ReadBlocking** (I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives data using a blocking method.
- void **SAI_ReadMultiChannelBlocking** (I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives multi channel data using a blocking method.
- static uint32_t **SAI_ReadData** (I2S_Type *base, uint32_t channel)

Reads data from the SAI FIFO.

Transactional

- void **SAI_TransferTxCreateHandle** (I2S_Type *base, sai_handle_t *handle, **sai_transfer_callback_t** callback, void *userData)

Initializes the SAI Tx handle.
- void **SAI_TransferRxCreateHandle** (I2S_Type *base, sai_handle_t *handle, **sai_transfer_callback_t** callback, void *userData)

Initializes the SAI Rx handle.
- void **SAI_TransferTxSetConfig** (I2S_Type *base, sai_handle_t *handle, **sai_transceiver_t** *config)

SAI transmitter transfer configurations.
- void **SAI_TransferRxSetConfig** (I2S_Type *base, sai_handle_t *handle, **sai_transceiver_t** *config)

SAI receiver transfer configurations.
- **status_t SAI_TransferTxSetFormat** (I2S_Type *base, sai_handle_t *handle, **sai_transfer_format_t** *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)

Configures the SAI Tx audio format.
- **status_t SAI_TransferRxSetFormat** (I2S_Type *base, sai_handle_t *handle, **sai_transfer_format_t** *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)

Configures the SAI Rx audio format.
- **status_t SAI_TransferSendNonBlocking** (I2S_Type *base, sai_handle_t *handle, **sai_transfer_t** *xfer)

Performs an interrupt non-blocking send transfer on SAI.
- **status_t SAI_TransferReceiveNonBlocking** (I2S_Type *base, sai_handle_t *handle, **sai_transfer_t** *xfer)

Performs an interrupt non-blocking receive transfer on SAI.
- **status_t SAI_TransferGetSendCount** (I2S_Type *base, sai_handle_t *handle, size_t *count)

Gets a set byte count.
- **status_t SAI_TransferGetReceiveCount** (I2S_Type *base, sai_handle_t *handle, size_t *count)

Gets a received byte count.
- void **SAI_TransferAbortSend** (I2S_Type *base, sai_handle_t *handle)

Aborts the current send.
- void **SAI_TransferAbortReceive** (I2S_Type *base, sai_handle_t *handle)

Aborts the current IRQ receive.

- void [SAI_TransferTerminateSend](#) (I2S_Type *base, sai_handle_t *handle)
Terminate all SAI send.
- void [SAI_TransferTerminateReceive](#) (I2S_Type *base, sai_handle_t *handle)
Terminate all SAI receive.
- void [SAI_TransferTxHandleIRQ](#) (I2S_Type *base, sai_handle_t *handle)
Tx interrupt handler.
- void [SAI_TransferRxHandleIRQ](#) (I2S_Type *base, sai_handle_t *handle)
Tx interrupt handler.

32.4.2 Data Structure Documentation

32.4.2.1 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.
- sai_bclk_source_t bclkSource
Bit Clock source.
- sai_master_slave_t masterSlave
Master or slave.

32.4.2.2 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`

32.4.2.3 `struct sai_fifo_t`

Data Fields

- `bool fifoContinueOneError`
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

32.4.2.4 `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`

32.4.2.5 `struct sai_frame_sync_t`

Data Fields

- `uint8_t frameSyncWidth`
frame sync width in number of bit clocks
- `bool frameSyncEarly`
TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame.
- `sai_clock_polarity_t frameSyncPolarity`
frame sync polarity

32.4.2.6 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

32.4.2.7 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.

32.4.2.8 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`

32.4.2.9 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.

32.4.3 Macro Definition Documentation

32.4.3.1 #define SAI_XFER_QUEUE_SIZE (4U)

32.4.4 Enumeration Type Documentation

32.4.4.1 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.

32.4.4.2 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

32.4.4.3 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.

32.4.4.4 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

32.4.4.5 enum sai_mono_stereo_t

Enumerator

kSAI_Stereo Stereo sound.

kSAI_MonoRight Only Right channel have sound.

kSAI_MonoLeft Only left channel have sound.

32.4.4.6 enum sai_data_order_t

Enumerator

kSAI_DataLSB LSB bit transferred first.

kSAI_DataMSB MSB bit transferred first.

32.4.4.7 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.

32.4.4.8 enum sai_sync_mode_t

Enumerator

kSAI_ModeAsync Asynchronous mode.

kSAI_ModeSync Synchronous mode (with receiver or transmit)

32.4.4.9 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.

32.4.4.10 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.

32.4.4.11 anonymous enum

Enumerator

kSAI_FIFOWarningDMAEnable FIFO warning caused by the DMA request.
kSAI_FIFORequestDMAEnable FIFO request caused by the DMA request.

32.4.4.12 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.

32.4.4.13 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.

32.4.4.14 enum sai_fifo_packing_t

Enumerator

kSAI_FifoPackingDisabled Packing disabled.

kSAI_FifoPacking8bit 8 bit packing enabled

kSAI_FifoPacking16bit 16bit packing enabled

32.4.4.15 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.

32.4.4.16 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.

32.4.4.17 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

32.4.4.18 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

32.4.4.19 enum sai_transceiver_type_t

Enumerator

kSAI_Transmitter sai transmitter
kSAI_Receiver sai receiver

32.4.4.20 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.

32.4.5 Function Documentation

32.4.5.1 void SAI_TxInit (I2S_Type * *base*, const sai_config_t * *config*)

Deprecated Do not use this function. It has been superceded by [SAI_Init](#)

Ungates the SAI clock, resets the module, and configures SAI Tx with a configuration structure. The configuration structure can be custom filled or set with default values by [SAI_TxGetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAIM module can cause a hard fault because the clock is not enabled.

Parameters

<i>base</i>	SAI base pointer
<i>config</i>	SAI configuration structure.

32.4.5.2 void SAI_RxInit (I2S_Type * *base*, const sai_config_t * *config*)

Deprecated Do not use this function. It has been superceded by [SAI_Init](#)

Ungates the SAI clock, resets the module, and configures the SAI Rx with a configuration structure. The configuration structure can be custom filled or set with default values by [SAI_RxGetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAI module can cause a hard fault because the clock is not enabled.

Parameters

<i>base</i>	SAI base pointer
<i>config</i>	SAI configuration structure.

32.4.5.3 void SAI_TxGetDefaultConfig (sai_config_t * *config*)

Deprecated Do not use this function. It has been superceded by [SAI_GetClassicI2SConfig](#), [SAI_GetLeft-JustifiedConfig](#) , [SAI_GetRightJustifiedConfig](#), [SAI_GetDSPConfig](#), [SAI_GetTDMConfig](#)

This API initializes the configuration structure for use in [SAI_TxConfig\(\)](#). The initialized structure can remain unchanged in [SAI_TxConfig\(\)](#), or it can be modified before calling [SAI_TxConfig\(\)](#). This is an example.

```
sai_config_t config;
SAI_TxGetDefaultConfig(&config);
```

Parameters

<i>config</i>	pointer to master configuration structure
---------------	---

32.4.5.4 void SAI_RxGetDefaultConfig (sai_config_t * *config*)

Deprecated Do not use this function. It has been superceded by [SAI_GetClassicI2SConfig](#), [SAI_GetLeftJustifiedConfig](#), [SAI_GetRightJustifiedConfig](#), [SAI_GetDSPConfig](#), [SAI_GetTDMConfig](#)

This API initializes the configuration structure for use in SAI_RxConfig(). The initialized structure can remain unchanged in SAI_RxConfig() or it can be modified before calling SAI_RxConfig(). This is an example.

```
sai_config_t config;
SAI_RxGetDefaultConfig(&config);
```

Parameters

<i>config</i>	pointer to master configuration structure
---------------	---

32.4.5.5 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.
-------------	-------------------

32.4.5.6 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.
-------------	-------------------

32.4.5.7 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
-------------	------------------

32.4.5.8 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
-------------	------------------

32.4.5.9 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.

32.4.5.10 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.

32.4.5.11 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.
--------------------	-------------------------------

32.4.5.12 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.

32.4.5.13 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.

32.4.5.14 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.

32.4.5.15 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.

32.4.5.16 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.

32.4.5.17 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.

32.4.5.18 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.

32.4.5.19 void SAI_TxSetFifoConfig (I2S_Type * *base*, sai_fifo_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	fifo configurations.

32.4.5.20 void SAI_RxSetFifoConfig (I2S_Type * *base*, sai_fifo_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	fifo configurations.

32.4.5.21 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.

32.4.5.22 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.

32.4.5.23 void SAI_TxSetSerialDataConfig (I2S_Type * *base*, sai_serial_data_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	serial data configurations.

32.4.5.24 void SAI_RxSetSerialDataConfig (I2S_Type * *base*, sai_serial_data_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	serial data configurations.

32.4.5.25 void SAI_TxSetConfig (I2S_Type * *base*, sai_transceiver_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	transmitter configurations.

32.4.5.26 void SAI_RxSetConfig (I2S_Type * *base*, sai_transceiver_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
-------------	-------------------

<i>config</i>	receiver configurations.
---------------	--------------------------

32.4.5.27 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.

32.4.5.28 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.

32.4.5.29 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.

32.4.5.30 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>frameSyncWidth</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.

32.4.5.31 void SAI_GetDSPConfig (*sai_transceiver_t * config, sai_frame_sync_len_t frameSyncWidth, sai_word_width_t bitWidth, sai_mono_stereo_t mode, uint32_t saiChannelMask*)

Note

DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
* SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth,
                   kSAI_Stereo, channelMask)
* config->frameSync.frameSyncEarly = true;
* SAI_TxSetConfig(base, config)
*
```

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
* SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth,
                   kSAI_Stereo, channelMask)
* SAI_TxSetConfig(base, config)
*
```

Parameters

<i>config</i>	transceiver configurations.
<i>frameSyncWidth</i>	length of frame sync.
<i>bitWidth</i>	audio data bitWidth.
<i>mode</i>	audio data channel.
<i>saiChannel-Mask</i>	mask value of the channel to enable.

32.4.5.32 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.

32.4.5.33 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

32.4.5.34 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.

32.4.5.35 static void SAI_RxClearStatusFlags (I2S_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	<p>State mask. It can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> • kSAI_WordStartFlag • kSAI_SyncErrorFlag • kSAI_FIFOErrorFlag

32.4.5.36 void SAI_TxSoftwareReset (I2S_Type * *base*, sai_reset_type_t *type*)

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

<i>base</i>	SAI base pointer
<i>type</i>	Reset type, FIFO reset or software reset

32.4.5.37 void SAI_RxSoftwareReset (I2S_Type * *base*, sai_reset_type_t *type*)

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

<i>base</i>	SAI base pointer
<i>type</i>	Reset type, FIFO reset or software reset

32.4.5.38 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.

32.4.5.39 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.

32.4.5.40 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

32.4.5.41 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
--------------	-----------------------

32.4.5.42 void SAI_TxSetBitClockPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

<i>base</i>	SAI base pointer
<i>polarity</i>	

32.4.5.43 void SAI_RxSetBitClockPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

<i>base</i>	SAI base pointer
<i>polarity</i>	

32.4.5.44 void SAI_TxSetFrameSyncPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

<i>base</i>	SAI base pointer
<i>polarity</i>	

32.4.5.45 void SAI_RxSetFrameSyncPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

<i>base</i>	SAI base pointer
<i>polarity</i>	

32.4.5.46 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.

32.4.5.47 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.

32.4.5.48 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.

32.4.5.49 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.

32.4.5.50 static void SAI_TxEnableInterrupts (I2S_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	<p>interrupt source The parameter can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

32.4.5.51 static void SAI_RxEnableInterrupts (I2S_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	<p>interrupt source The parameter can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

32.4.5.52 static void SAI_TxDisableInterrupts (I2S_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	<p>interrupt source The parameter can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

**32.4.5.53 static void SAI_RxDisableInterrupts (I2S_Type * *base*, uint32_t *mask*)
[inline], [static]**

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	<p>interrupt source The parameter can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

**32.4.5.54 static void SAI_TxEnableDMA (I2S_Type * *base*, uint32_t *mask*, bool *enable*)
[inline], [static]**

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	<p>DMA source The parameter can be combination of the following sources if defined.</p> <ul style="list-style-type: none"> • kSAI_FIFOWarningDMAEnable • kSAI_FIFORequestDMAEnable
<i>enable</i>	True means enable DMA, false means disable DMA.

**32.4.5.55 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.

32.4.5.56 static uint32_t SAI_TxGetDataRegisterAddress (I2S_Type * *base*, uint32_t *channel*) [inline], [static]

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Which data channel used.

Returns

data register address.

32.4.5.57 static uint32_t SAI_RxGetDataRegisterAddress (I2S_Type * *base*, uint32_t *channel*) [inline], [static]

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Which data channel used.

Returns

data register address.

32.4.5.58 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.

32.4.5.59 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.

32.4.5.60 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.

32.4.5.61 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.

32.4.5.62 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.

32.4.5.63 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.

32.4.5.64 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.

32.4.5.65 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.

**32.4.5.66 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

32.4.5.67 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.

32.4.5.68 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.

32.4.5.69 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.

32.4.5.70 status_t SAI_TransferTxSetFormat (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

Deprecated Do not use this function. It has been superceded by [SAI_TransferTxSetConfig](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>format</i>	Pointer to the SAI audio data format structure.
<i>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 <i>masterClockHz</i> in <i>format</i> .

Returns

Status of this function. Return value is the *status_t*.

32.4.5.71 status_t SAI_TransferRxSetFormat (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

Deprecated Do not use this function. It has been superceded by [SAI_TransferRxSetConfig](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>format</i>	Pointer to the SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If a bit clock source is a master clock, this value should equal the masterClockHz in format.

Returns

Status of this function. Return value is one of status_t.

32.4.5.72 status_t SAI_TransferSendNonBlocking (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_t * *xfer*)

Note

This API returns immediately after the transfer initiates. Call the SAI_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.
<i>xfer</i>	Pointer to the sai_transfer_t structure.

Return values

<i>kStatus_Success</i>	Successfully started the data receive.
<i>kStatus_SAI_TxBusy</i>	Previous receive still not finished.
<i>kStatus_InvalidArgument</i>	The input parameter is invalid.

32.4.5.73 status_t SAI_TransferReceiveNonBlocking (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_t * *xfer*)

Note

This API returns immediately after the transfer initiates. Call the SAI_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus__SAI_Busy, the transfer is finished.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.
<i>xfer</i>	Pointer to the sai_transfer_t structure.

Return values

<i>kStatus_Success</i>	Successfully started the data receive.
<i>kStatus_SAI_RxBusy</i>	Previous receive still not finished.
<i>kStatus_InvalidArgument</i>	The input parameter is invalid.

32.4.5.74 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 sai_handle_t structure which stores the transfer state.
<i>count</i>	Bytes count sent.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferIn-Progress</i>	There is not a non-blocking transaction currently in progress.

32.4.5.75 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_NoTransferIn-Progress</i>	There is not a non-blocking transaction currently in progress.

32.4.5.76 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.

32.4.5.77 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.

32.4.5.78 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.

32.4.5.79 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.

32.4.5.80 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.

32.4.5.81 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.

32.5 SAI EDMA Driver

32.5.1 Overview

Data Structures

- struct `sai_edma_handle_t`
SAI DMA transfer handle, users should not touch the content of the handle. [More...](#)

Typedefs

- typedef void(* `sai_edma_callback_t`)(I2S_Type *base, sai_edma_handle_t *handle, `status_t` status, void *userData)
SAI eDMA transfer callback function for finish and error.

Driver version

- #define `FSL_SAI_EDMA_DRIVER_VERSION` (`MAKE_VERSION(2, 5, 0)`)
Version 2.5.0.

eDMA Transactional

- void `SAI_TransferTxCreateHandleEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_edma_callback_t` callback, void *userData, `edma_handle_t` *txDmaHandle)
Initializes the SAI eDMA handle.
- void `SAI_TransferRxCreateHandleEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_edma_callback_t` callback, void *userData, `edma_handle_t` *rxDmaHandle)
Initializes the SAI Rx eDMA handle.
- void `SAI_TransferTxSetFormatEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_transfer_format_t` *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
Configures the SAI Tx audio format.
- void `SAI_TransferRxSetFormatEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_transfer_format_t` *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
Configures the SAI Rx audio format.
- void `SAI_TransferTxSetConfigEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_transceiver_t` *saiConfig)
Configures the SAI Tx.
- void `SAI_TransferRxSetConfigEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_transceiver_t` *saiConfig)
Configures the SAI Rx.
- `status_t SAI_TransferSendEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_transfer_t` *xfer)
Performs a non-blocking SAI transfer using DMA.
- `status_t SAI_TransferReceiveEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_transfer_t` *xfer)

- *Performs a non-blocking SAI receive using eDMA.*
- **status_t SAI_TransferSendLoopEDMA** (I2S_Type *base, sai_edma_handle_t *handle, sai_transfer_t *xfer, uint32_t loopTransferCount)
 - Performs a non-blocking SAI loop transfer using eDMA.*
- **status_t SAI_TransferReceiveLoopEDMA** (I2S_Type *base, sai_edma_handle_t *handle, sai_transfer_t *xfer, uint32_t loopTransferCount)
 - Performs a non-blocking SAI loop transfer using eDMA.*
- **void SAI_TransferTerminateSendEDMA** (I2S_Type *base, sai_edma_handle_t *handle)
 - Terminate all SAI send.*
- **void SAI_TransferTerminateReceiveEDMA** (I2S_Type *base, sai_edma_handle_t *handle)
 - Terminate all SAI receive.*
- **void SAI_TransferAbortSendEDMA** (I2S_Type *base, sai_edma_handle_t *handle)
 - Aborts a SAI transfer using eDMA.*
- **void SAI_TransferAbortReceiveEDMA** (I2S_Type *base, sai_edma_handle_t *handle)
 - Aborts a SAI receive using eDMA.*
- **status_t SAI_TransferGetSendCountEDMA** (I2S_Type *base, sai_edma_handle_t *handle, size_t *count)
 - Gets byte count sent by SAI.*
- **status_t SAI_TransferGetReceiveCountEDMA** (I2S_Type *base, sai_edma_handle_t *handle, size_t *count)
 - Gets byte count received by SAI.*
- **uint32_t SAI_TransferGetValidTransferSlotsEDMA** (I2S_Type *base, sai_edma_handle_t *handle)
 - Gets valid transfer slot.*

32.5.2 Data Structure Documentation

32.5.2.1 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`

32.5.3 Function Documentation

32.5.3.1 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.

32.5.3.2 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.

32.5.3.3 void SAI_TransferTxSetFormatEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

Deprecated Do not use this function. It has been superceded by [SAI_TransferTxSetConfigEDMA](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>format</i>	Pointer to SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If bit clock source is master clock, this value should equals to masterClockHz in format.

Return values

<i>kStatus_Success</i>	Audio format set successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

32.5.3.4 void SAI_TransferRxSetFormatEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

Deprecated Do not use this function. It has been superceded by [SAI_TransferRxSetConfigEDMA](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>format</i>	Pointer to SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If a bit clock source is the master clock, this value should equal to <i>masterClockHz</i> in format.

Return values

<i>kStatus_Success</i>	Audio format set successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

32.5.3.5 void SAI_TransferTxSetConfigEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transceiver_t * *saiConfig*)

Note

SAI eDMA supports data transfer in multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of *sai_transceiver_t* with the corresponding values of *_sai_channel_mask* enum, enable the FIFO Combine mode by assigning *kSAI_FifoCombineModeEnabledOnWrite* to the *fifoCombine* member of *sai_fifo_combine_t* which is a member of *sai_transceiver_t*. This is an example of multi-channel data transfer configuration step.

```
*   sai_transceiver_t config;
*   SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits,
*                           kSAI_Stereo, kSAI_Channel0Mask|kSAI_Channel1Mask);
*   config fifo fifoCombine = kSAI_FifoCombineModeEnabledOnWrite
*   ;
*   SAI_TransferTxSetConfigEDMA(I2S0, &edmaHandle, &config);
*
```

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>saiConfig</i>	sai configurations.

32.5.3.6 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.

32.5.3.7 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.

32.5.3.8 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.
-----------------------	-----------------------------

32.5.3.9 status_t SAI_TransferSendLoopEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_t * *xfer*, uint32_t *loopTransferCount*)

Note

This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size.

Once the loop transfer start, application can use function SAI_TransferAbortSendEDMA to stop the loop transfer.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>xfer</i>	Pointer to the DMA transfer structure, should be a array with elements counts ≥ 1 (loopTransferCount).
<i>loopTransfer-Count</i>	the counts of xfer array.

Return values

<i>kStatus_Success</i>	Start a SAI eDMA send successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

32.5.3.10 status_t SAI_TransferReceiveLoopEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_t * *xfer*, uint32_t *loopTransferCount*)

Note

This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size.

Once the loop transfer start, application can use function SAI_TransferAbortReceiveEDMA to stop the loop transfer.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>xfer</i>	Pointer to the DMA transfer structure, should be a array with elements counts ≥ 1 (loopTransferCount).
<i>loopTransfer-Count</i>	the counts of xfer array.

Return values

<i>kStatus_Success</i>	Start a SAI eDMA receive successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

32.5.3.11 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.

32.5.3.12 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.

32.5.3.13 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.

32.5.3.14 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.

32.5.3.15 status_t SAI_TransferGetSendCountEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, size_t * *count*)

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>count</i>	Bytes count sent by SAI.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferIn-Progress</i>	There is no non-blocking transaction in progress.

32.5.3.16 status_t SAI_TransferGetReceiveCountEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, size_t * *count*)

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI eDMA handle pointer.
<i>count</i>	Bytes count received by SAI.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferIn-Progress</i>	There is no non-blocking transaction in progress.

32.5.3.17 `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 33

SNVS: Secure Non-Volatile Storage

33.1 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](#)

33.2 Secure Non-Volatile Storage High-Power

33.2.1 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 << (SNVS_HPSVR_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, 1)`)
Version 2.3.1.

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.
- void `SNVS_HP_RTC_TimeSynchronize` (SNVS_Type *base)
The function synchronizes RTC counter value with SRTC.

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)

33.2.2 Data Structure Documentation

33.2.2.1 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`

33.2.2.2 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.

33.2.3 Macro Definition Documentation

33.2.3.1 #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.

33.2.4 Enumeration Type Documentation

33.2.4.1 enum snvs_hp_interrupts_t

Enumerator

- kSNVS_RTC_AlarmInterrupt* RTC time alarm.
- kSNVS_RTC_PeriodicInterrupt* RTC periodic interrupt.

33.2.4.2 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.

33.2.4.3 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.

33.2.4.4 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.

33.2.5 Function Documentation

33.2.5.1 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
-------------	------------------------------

33.2.5.2 void SNVS_HP_Deinit (SNVS_Type * *base*)

Parameters

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

33.2.5.3 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.

33.2.5.4 void SNVS_HP_RTC_Deinit (SNVS_Type * *base*)

Parameters

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

33.2.5.5 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.
---------------	---

33.2.5.6 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

33.2.5.7 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.

33.2.5.8 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

33.2.5.9 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.

33.2.5.10 void SNVS_HP_RTC_TimeSynchronize (SNVS_Type * *base*)

Parameters

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

33.2.5.11 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 :: _snvs_hp_interrupts_t

33.2.5.12 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 :: _snvs_hp_interrupts_t

33.2.5.13 `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

33.2.5.14 `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

33.2.5.15 `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 :: _snvs_hp_status_flags_t

33.2.5.16 `static void SNVS_HP_RTC_StartTimer (SNVS_Type * base) [inline], [static]`

Parameters

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

33.2.5.17 static void SNVS_HP_RTC_StopTimer (SNVS_Type * *base*) [inline], [static]

Parameters

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

33.2.5.18 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.

33.2.5.19 static void SNVS_HP_ProgramZeroizableMasterKey (SNVS_Type * *base*) [inline], [static]

Parameters

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

33.2.5.20 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
-------------	------------------------------

33.2.5.21 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
-------------	------------------------------

33.2.5.22 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
-------------	------------------------------

33.2.5.23 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

33.2.5.24 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
-------------	------------------------------

33.2.5.25 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.

33.2.5.26 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.

33.2.5.27 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.

33.2.5.28 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
-------------	------------------------------

33.2.5.29 static uint32_t SNVS_HP_GetHighAssuranceCounter (SNVS_Type * *base*) [inline], [static]

Parameters

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

Returns

HAC currnet value.

33.2.5.30 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
-------------	------------------------------

33.2.5.31 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
-------------	------------------------------

33.2.5.32 static uint32_t SNVS_HP_GetStatusFlags (SNVS_Type * *base*) [inline], [static]

The flags are returned as the OR'ed value f the enumeration :: _snvs_hp_status_flags_t.

Parameters

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

Returns

The OR'ed value of status flags.

33.2.5.33 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.

33.2.5.34 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.

33.2.5.35 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.

33.3 Secure Non-Volatile Storage Low-Power

33.3.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Secure Non-Volatile Storage Low-Power (S-NVS-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_M-ASK }
List of SNVS_LP interrupts.
- enum `snvs_lp_srtc_status_flags_t` { `kSNVS_SRTC_AlarmInterruptFlag` = SNVS_LPSR_LPTA-MASK }
List of SNVS_LP flags.
- enum `snvs_lp_external_tamper_status_t`
List of SNVS_LP external tampers status.
- enum `snvs_lp_external_tamper_polarity_t`
SNVS_LP external tamper polarity.
- enum `snvs_lp_zmk_program_mode_t` {
`kSNVS_ZMKSoftwareProgram`,
`kSNVS_ZMKHardwareProgram` }
SNVS_LP Zeroizable Master Key programming mode.
- 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, 3))
Version 2.4.3.

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** *alarmTime)
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.

33.3.2 Data Structure Documentation

33.3.2.1 `struct snvs_lp_passive_tamper_t`

33.3.2.2 `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`

33.3.2.3 `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.

33.3.3 Enumeration Type Documentation

33.3.3.1 enum snvs_lp_srtc_interrupts_t

Enumerator

kSNVS_SRTC_AlarmInterrupt SRTC time alarm.

33.3.3.2 enum snvs_lp_srtc_status_flags_t

Enumerator

kSNVS_SRTC_AlarmInterruptFlag SRTC time alarm flag.

33.3.3.3 enum snvs_lp_zmk_program_mode_t

Enumerator

kSNVS_ZMKSoftwareProgram Software programming mode.

kSNVS_ZMKHardwareProgram Hardware programming mode.

33.3.3.4 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 OTPMK and ZMK.

33.3.4 Function Documentation

33.3.4.1 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
-------------	------------------------------

33.3.4.2 void SNVS_LP_Deinit (SNVS_Type * *base*)

Parameters

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

33.3.4.3 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.

33.3.4.4 void SNVS_LP_SRTC_Deinit (SNVS_Type * *base*)

Parameters

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

33.3.4.5 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.
---------------	---

33.3.4.6 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

33.3.4.7 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.

33.3.4.8 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

33.3.4.9 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.

33.3.4.10 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 :: _-snvs_lp_srtc_interrupts

33.3.4.11 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 :: _-snvs_lp_srtc_interrupts

33.3.4.12 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

33.3.4.13 **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

33.3.4.14 **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 :: _-snvs_lp_srtc_status_flags

33.3.4.15 **static void SNVS_LP_SRTC_StartTimer (SNVS_Type * *base*) [inline], [static]**

Parameters

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

33.3.4.16 **static void SNVS_LP_SRTC_StopTimer (SNVS_Type * *base*) [inline], [static]**

Parameters

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

33.3.4.17 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.
---------------	---

33.3.4.18 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.

33.3.4.19 uint64_t SNVS_LP_GetMonotonicCounter (**SNVS_Type** * *base*)

Parameters

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

Returns

Current Monotonic Counter value.

33.3.4.20 static void SNVS_LP_IncreaseMonotonicCounter (**SNVS_Type** * *base*) [inline], [static]

Increase the Monotonic Counter by 1.

Parameters

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

33.3.4.21 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.

33.3.4.22 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.

33.3.4.23 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.

33.3.4.24 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.

33.3.4.25 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.

33.3.4.26 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 34

SPDIF: Sony/Philips Digital Interface

34.1 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.

34.2 Typical use case

34.2.1 SPDIF Send/receive using an interrupt method

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

34.2.2 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_RxDPLLocked = 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_ValidateNoGood = 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_RxDPLLocked` = SPDIF_SIE_LOCK_MASK,
 `kSPDIF_TxFIFOError` = SPDIF_SIE_TXUNOV_MASK,
 `kSPDIF_TxFIFOResync` = SPDIF_SIE_TXRESYN_MASK,
 `kSPDIF_RxControlChannelChange` = SPDIF_SIE_CNEW_MASK,
 `kSPDIF_ValidityFlagNoGood` = 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 **SPDIFGetInstance** (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.

34.3 Data Structure Documentation

34.3.1 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**

34.3.2 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`

34.3.3 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.

34.4 Macro Definition Documentation

34.4.1 #define SPDIF_XFER_QUEUE_SIZE (4U)

34.5 Enumeration Type Documentation

34.5.1 anonymous enum

Enumerator

kStatus_SPDIF_RxPLLLocked 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_ValidateNoGood SPDIF validate 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.

34.5.2 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.

34.5.3 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.

34.5.4 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.

34.5.5 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.

34.5.6 enum spdif_tx_source_t

Enumerator

kSPDIF_txFromReceiver Tx data directly through SPDIF receiver.

kSPDIF_txNormal Normal operation, data from processor.

34.5.7 enum spdif_validity_config_t

Enumerator

kSPDIF_validityFlagAlwaysSet Outgoing validity flags always set.

kSPDIF_validityFlagAlwaysClear Outgoing validity flags always clear.

34.5.8 anonymous enum

Enumerator

kSPDIF_RxDPLLocked 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

34.5.9 anonymous enum

Enumerator

kSPDIF_RxDMAEnable Rx FIFO full.
kSPDIF_TxDMAEnable Tx FIFO empty.

34.6 Function Documentation

34.6.1 void SPDIF_Init (**SPDIF_Type** * *base*, **const spdif_config_t** * *config*)

Ungates 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.
---------------	--------------------------------

34.6.2 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
---------------	---

34.6.3 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
-------------	--------------------

34.6.4 uint32_t SPDIFGetInstance (*SPDIF_Type* * *base*)

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

34.6.5 static void SPDIF_TxFIFOReset (*SPDIF_Type* * *base*) [inline], [static]

This function makes Tx FIFO in reset mode.

Parameters

<i>base</i>	SPDIF base pointer
-------------	--------------------

34.6.6 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
-------------	--------------------

34.6.7 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.

34.6.8 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.

34.6.9 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.

34.6.10 static void SPDIF_ClearStatusFlags (**SPDIF_Type** * *base*, **uint32_t** *mask*) [inline], [static]

Parameters

<i>base</i>	SPDIF base pointer
<i>mask</i>	<p>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:</p> <ul style="list-style-type: none"> • <code>kSPDIF_UChannelReceiveRegisterFull</code> • <code>kSPDIF_QChannelReceiveRegisterFull</code> • <code>kSPDIF_TxFIFOEmpty</code> • <code>kSPDIF_RxFIFOFull</code>

34.6.11 static void SPDIF_EnableInterrupts (**SPDIF_Type** * *base*, **uint32_t** *mask*) [inline], [static]

Parameters

<i>base</i>	SPDIF base pointer
<i>mask</i>	<p>interrupt source The parameter can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> • <code>kSPDIF_WordStartInterruptEnable</code> • <code>kSPDIF_SyncErrorInterruptEnable</code> • <code>kSPDIF_FIFOWarningInterruptEnable</code> • <code>kSPDIF_FIFORequestInterruptEnable</code> • <code>kSPDIF_FIFOErrorInterruptEnable</code>

34.6.12 static void SPDIF_DisableInterrupts (**SPDIF_Type** * *base*, **uint32_t** *mask*) [inline], [static]

Parameters

<i>base</i>	SPDIF base pointer
<i>mask</i>	<p>interrupt source The parameter can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> • kSPDIF_WordStartInterruptEnable • kSPDIF_SyncErrorInterruptEnable • kSPDIF_FIFOWarningInterruptEnable • kSPDIF_FIFOResponseInterruptEnable • kSPDIF_FIFOErrorInterruptEnable

34.6.13 static void SPDIF_EnableDMA (**SPDIF_Type** * *base*, **uint32_t** *mask*, **bool enable**) [inline], [static]

Parameters

<i>base</i>	SPDIF base pointer
<i>mask</i>	<p>SPDIF DMA enable mask, The parameter can be a combination of the following sources if defined</p> <ul style="list-style-type: none"> • kSPDIF_RxDMAEnable • kSPDIF_TxDMAEnable
<i>enable</i>	True means enable DMA, false means disable DMA.

34.6.14 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.

**34.6.15 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.

34.6.16 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.

34.6.17 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.

34.6.18 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.

34.6.19 **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.

34.6.20 **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.

34.6.21 **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.

34.6.22 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.

34.6.23 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.

34.6.24 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.

34.6.25 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.

34.6.26 static uint32_t SPDIF_ReadLeftData (SPDIF_Type * *base*) [inline], [static]

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

Returns

Data in SPDIF FIFO.

34.6.27 static uint32_t SPDIF_ReadRightData (SPDIF_Type * *base*) [inline], [static]

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

Returns

Data in SPDIF FIFO.

34.6.28 static uint32_t SPDIF_ReadChannelStatusHigh (SPDIF_Type * *base*) [inline], [static]

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

Returns

Data in SPDIF FIFO.

**34.6.29 static uint32_t SPDIF_ReadChannelStatusLow (SPDIF_Type * *base*)
[inline], [static]**

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

Returns

Data in SPDIF FIFO.

**34.6.30 static uint32_t SPDIF_ReadQChannel (SPDIF_Type * *base*) [inline],
[static]**

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

Returns

Data in SPDIF FIFO.

**34.6.31 static uint32_t SPDIF_ReadUChannel (SPDIF_Type * *base*) [inline],
[static]**

Parameters

<i>base</i>	SPDIF base pointer.
-------------	---------------------

Returns

Data in SPDIF FIFO.

34.6.32 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

34.6.33 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.

34.6.34 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.

34.6.35 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 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_RxBusy</i>	Previous receive still not finished.
<i>kStatus_InvalidArgument</i>	The input parameter is invalid.

34.6.36 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 spdif_handle_t structure which stores the transfer state.
<i>count</i>	Bytes count sent.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferIn-Progress</i>	There is not a non-blocking transaction currently in progress.

34.6.37 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 spdif_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_NoTransferIn-Progress</i>	There is not a non-blocking transaction currently in progress.

34.6.38 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 spdif_handle_t structure which stores the transfer state.

34.6.39 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.

34.6.40 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.

34.6.41 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.

34.7 SPDIF eDMA Driver

34.7.1 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.

34.7.2 Data Structure Documentation

34.7.2.1 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**

34.7.2.2 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`

34.7.3 Function Documentation

**34.7.3.1 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

<code>base</code>	SPDIF base pointer.
<code>handle</code>	SPDIF eDMA handle pointer.
<code>base</code>	SPDIF peripheral base address.
<code>callback</code>	Pointer to user callback function.
<code>userData</code>	User parameter passed to the callback function.
<code>dmaLeftHandle</code>	eDMA handle pointer for left channel, this handle shall be static allocated by users.
<code>dmaRightHandle</code>	eDMA handle pointer for right channel, this handle shall be static allocated by users.

```
34.7.3.2 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>dmaRightHandle</i>	eDMA handle pointer for right channel, this handle shall be static allocated by users.

34.7.3.3 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 SPIDF_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.

34.7.3.4 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 SPIDF_GetReceiveRemainingBytes 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.

34.7.3.5 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.

34.7.3.6 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.

34.7.3.7 **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_NoTransferIn-Progress</i>	There is no non-blocking transaction in progress.

34.7.3.8 **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_NoTransferIn-Progress</i>	There is no non-blocking transaction in progress.

Chapter 35

SRC: System Reset Controller Driver

35.1 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_TemperatureSensorResetFlag` = SRC_SRSR_TS_R_MASK,
 `kSRC_Wdog3ResetFlag` = SRC_SRSR_WDOG3_RST_B_MASK,
 `kSRC_JTAGSystemResetFlag`,
 `kSRC_JTAGSoftwareResetFlag` = SRC_SRSR_SJC_MASK,
 `kSRC_JTAGGeneratedResetFlag` = SRC_SRSR_JTAG_MASK,
 `kSRC_WatchdogResetFlag` = SRC_SRSR_WDOG_MASK,
 `kSRC_IppUserResetFlag` = SRC_SRSR_IPP_USER_RESET_B_MASK,
 `kSRC_CsuResetFlag` = SRC_SRSR_CSU_RESET_B_MASK,
 `kSRC_CoreLockupResetFlag` = SRC_SRSR_LOCKUP_MASK,
 `kSRC_IppResetPinFlag` = SRC_SRSR_IPP_RESET_B_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_EnableWDOG3Reset` (SRC_Type *base, bool enable)
 Enable the WDOG3 reset.
- static void `SRC_EnableCoreDebugResetAfterPowerGate` (SRC_Type *base, bool enable)
 Debug reset would be asserted after power gating event.
- static void `SRC_DoSoftwareResetARMCore0` (SRC_Type *base)
 Do software reset the ARM core0 only.
- static bool `SRC_GetSoftwareResetARMCore0Done` (SRC_Type *base)
 Check if the software for ARM core0 is done.
- static void `SRC_EnableWDOGReset` (SRC_Type *base, bool enable)
 Enable the WDOG Reset in SRC.

- static void [SRC_EnableLockupReset](#) (SRC_Type *base, bool enable)
Enable the lockup 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 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.

35.2 Macro Definition Documentation

35.2.1 #define FSL_SRC_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

35.3 Enumeration Type Documentation

35.3.1 enum _src_reset_status_flags

Enumerator

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_Wdog3ResetFlag IC Watchdog3 Time-out reset. Indicates whether the reset was the result of the watchdog3 time-out event.

kSRC_JTAGSystemResetFlag Indicates whether the reset was the result of software reset form JT-AG.

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.

kSRC_IppUserResetFlag Indicates whether the reset was the result of the ipp_user_reset_b qualified reset.

kSRC_CsuResetFlag Indicates whether the reset was the result of the csu_reset_b input.

kSRC_CoreLockupResetFlag Indicates a reset has been caused by the ARM core indication of a LOCKUP event.

kSRC_IppResetPinFlag Indicates whether reset was the result of `ipp_reset_b` pin (Power-up sequence).

35.3.2 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.

35.4 Function Documentation

35.4.1 static void SRC_EnableWDOG3Reset (`SRC_Type * base`, `bool enable`) [inline], [static]

The WDOG3 reset is enabled by default.

Parameters

<i>base</i>	SRC peripheral base address.
<i>enable</i>	Enable the reset or not.

35.4.2 static void SRC_EnableCoreDebugResetAfterPowerGate (`SRC_Type * base`, `bool enable`) [inline], [static]

Parameters

<i>base</i>	SRC peripheral base address.
<i>enable</i>	Enable the reset or not.

35.4.3 static void SRC_DoSoftwareResetARMCore0 (`SRC_Type * base`) [inline], [static]

Parameters

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

35.4.4 static bool SRC_GetSoftwareResetARMCore0Done (SRC_Type * *base*) [inline], [static]

Parameters

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

Returns

If the reset is done.

35.4.5 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.

35.4.6 static void SRC_EnableLockupReset (SRC_Type * *base*, bool *enable*) [inline], [static]

Parameters

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

<i>enable</i>	Enable the reset or not.
---------------	--------------------------

35.4.7 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.

35.4.8 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.

35.4.9 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](#).

35.4.10 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 .

35.4.11 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.

35.4.12 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 36

TEMPMON: Temperature Monitor Module

36.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Temperature Monitor Module (TEMPMON) module of MCUXpresso SDK devices.

36.2 TEMP MON: Temperature Monitor Module

36.2.1 TEMP MON Operations

The function `TEMPMON_Init()` will initialize the TEMP MON peripheral operation.

The function `TEMPMON_Deinit()` will deinitialize the TEMP MON 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`
TEMP MON temperature structure. More...

Enumerations

- enum `tempmon_alarm_mode` {
 `kTEMPMON_HighAlarmMode` = 0U,
 `kTEMPMON_PanicAlarmMode` = 1U,
 `kTEMPMON_LowAlarmMode` = 2U }
TEMP MON alarm mode.

Functions

- void `TEMPMON_Init` (TEMPMON_Type *base, const `tempmon_config_t` *config)

- `void TEMPMON_Deinit (TEMPMON_Type *base)`
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.

36.3 Data Structure Documentation

36.3.1 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`

36.4 Macro Definition Documentation

36.4.1 #define FSL_TEMPMON_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

36.5 Enumeration Type Documentation

36.5.1 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.

36.6 Function Documentation

36.6.1 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.

36.6.2 void TEMPMON_Deinit (**TEMPMON_Type** * *base*)

Parameters

<i>base</i>	TEMPMON base pointer
-------------	----------------------

36.6.3 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.
---------------	---------------------------------------

36.6.4 static void TEMPMON_StartMeasure (**TEMPMON_Type** * *base*) [inline], [static]

Parameters

<i>base</i>	TEMPMON base pointer.
-------------	-----------------------

36.6.5 static void TEMPMON_StopMeasure (**TEMPMON_Type** * *base*) [inline], [static]

Parameters

<i>base</i>	TEMPMON base pointer
-------------	----------------------

36.6.6 float TEMPMON_GetCurrentTemperature (**TEMPMON_Type** * *base*)

Parameters

<i>base</i>	TEMPMON base pointer
-------------	----------------------

Returns

current temperature with degrees Celsius.

36.6.7 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 37

TRNG: True Random Number Generator

37.1 Overview

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.

37.2 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.

37.3 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

Data Structures

- struct `trng_statistical_check_limit_t`
Data structure for definition of statistical check limits. [More...](#)
- struct `trng_config_t`
Data structure for the TRNG initialization. [More...](#)

Enumerations

- enum `trng_sample_mode_t` {
 `kTRNG_SampleModeVonNeumann` = 0U,
 `kTRNG_SampleModeRaw` = 1U,
 `kTRNG_SampleModeVonNeumannRaw` }
TRNG sample mode.

- enum `trng_clock_mode_t` {

 `kTRNG_ClockModeRingOscillator` = 0U,

 `kTRNG_ClockModeSystem` = 1U }

TRNG clock mode.
- enum `trng_ring_osc_div_t` {

 `kTRNG_RingOscDiv0` = 0U,

 `kTRNG_RingOscDiv2` = 1U,

 `kTRNG_RingOscDiv4` = 2U,

 `kTRNG_RingOscDiv8` = 3U }

TRNG ring oscillator divide.

Functions

- `status_t TRNG_GetDefaultConfig (trng_config_t *userConfig)`

Initializes the user configuration structure to default values.
- `status_t TRNG_Init (TRNG_Type *base, const trng_config_t *userConfig)`

Initializes the TRNG.
- `void TRNG_Deinit (TRNG_Type *base)`

Shuts down the TRNG.
- `status_t TRNG_GetRandomData (TRNG_Type *base, void *data, size_t dataSize)`

Gets random data.

Driver version

- `#define FSL_TRNG_DRIVER_VERSION (MAKE_VERSION(2, 0, 13))`

TRNG driver version 2.0.13.

37.4 Data Structure Documentation

37.4.1 struct `trng_statistical_check_limit_t`

Used by `trng_config_t`.

Data Fields

- `uint32_t maximum`

Maximum limit.
- `uint32_t minimum`

Minimum limit.

Field Documentation

- (1) `uint32_t trng_statistical_check_limit_t::maximum`
- (2) `uint32_t trng_statistical_check_limit_t::minimum`

37.4.2 struct trng_config_t

This structure initializes the TRNG by calling the `TRNG_Init()` function. It contains all TRNG configurations.

Data Fields

- `bool lock`
Disable programmability of TRNG registers.
- `trng_clock_mode_t clockMode`
Clock mode used to operate TRNG.
- `trng_ring_osc_div_t ringOscDiv`
Ring oscillator divide used by TRNG.
- `trng_sample_mode_t sampleMode`
Sample mode of the TRNG ring oscillator.
- `uint16_t entropyDelay`
Entropy Delay.
- `uint16_t sampleSize`
Sample Size.
- `uint16_t sparseBitLimit`
Sparse Bit Limit which defines the maximum number of consecutive samples that may be discarded before an error is generated.
- `uint8_t retryCount`
Retry count.
- `uint8_t longRunMaxLimit`
Largest allowable number of consecutive samples of all 1, or all 0, that is allowed during the Entropy generation.
- `trng_statistical_check_limit_t monobitLimit`
Maximum and minimum limits for statistical check of number of ones/zero detected during entropy generation.
- `trng_statistical_check_limit_t runBit1Limit`
Maximum and minimum limits for statistical check of number of runs of length 1 detected during entropy generation.
- `trng_statistical_check_limit_t runBit2Limit`
Maximum and minimum limits for statistical check of number of runs of length 2 detected during entropy generation.
- `trng_statistical_check_limit_t runBit3Limit`
Maximum and minimum limits for statistical check of number of runs of length 3 detected during entropy generation.
- `trng_statistical_check_limit_t runBit4Limit`
Maximum and minimum limits for statistical check of number of runs of length 4 detected during entropy generation.
- `trng_statistical_check_limit_t runBit5Limit`
Maximum and minimum limits for statistical check of number of runs of length 5 detected during entropy generation.
- `trng_statistical_check_limit_t runBit6PlusLimit`
Maximum and minimum limits for statistical check of number of runs of length 6 or more detected during entropy generation.
- `trng_statistical_check_limit_t pokerLimit`

- *Maximum and minimum limits for statistical check of "Poker Test".*
- **trng_statistical_check_limit_t frequencyCountLimit**
Maximum and minimum limits for statistical check of entropy sample frequency count.

Field Documentation

- (1) **bool trng_config_t::lock**
- (2) **trng_clock_mode_t trng_config_t::clockMode**
- (3) **trng_ring_osc_div_t trng_config_t::ringOscDiv**
- (4) **trng_sample_mode_t trng_config_t::sampleMode**
- (5) **uint16_t trng_config_t::entropyDelay**

Defines the length (in system clocks) of each Entropy sample taken.

- (6) **uint16_t trng_config_t::sampleSize**

Defines the total number of Entropy samples that will be taken during Entropy generation.

- (7) **uint16_t trng_config_t::sparseBitLimit**

This limit is used only for during von Neumann sampling (enabled by TRNG_HAL_SetSampleMode()). Samples are discarded if two consecutive raw samples are both 0 or both 1. If this discarding occurs for a long period of time, it indicates that there is insufficient Entropy.

- (8) **uint8_t trng_config_t::retryCount**

It defines the number of times a statistical check may fails during the TRNG Entropy Generation before generating an error.

- (9) **uint8_t trng_config_t::longRunMaxLimit**
- (10) **trng_statistical_check_limit_t trng_config_t::monobitLimit**
- (11) **trng_statistical_check_limit_t trng_config_t::runBit1Limit**
- (12) **trng_statistical_check_limit_t trng_config_t::runBit2Limit**
- (13) **trng_statistical_check_limit_t trng_config_t::runBit3Limit**
- (14) **trng_statistical_check_limit_t trng_config_t::runBit4Limit**
- (15) **trng_statistical_check_limit_t trng_config_t::runBit5Limit**
- (16) **trng_statistical_check_limit_t trng_config_t::runBit6PlusLimit**
- (17) **trng_statistical_check_limit_t trng_config_t::pokerLimit**

(18) `trng_statistical_check_limit_t trng_config_t::frequencyCountLimit`

37.5 Macro Definition Documentation

37.5.1 `#define FSL_TRNG_DRIVER_VERSION (MAKE_VERSION(2, 0, 13))`

Current version: 2.0.13

Change log:

- version 2.0.13
 - After deepsleep it might return error, added clearing bits in `TRNG_GetRandomData()` and generating new entropy.
 - Modified reloading entropy in `TRNG_GetRandomData()`, for some data length it doesn't reloading entropy correctly.
- version 2.0.12
 - For KW34A4_SERIES, KW35A4_SERIES, KW36A4_SERIES set `TRNG_USER_CONFIG_DEFAULT_OSC_DIV` to `kTRNG_RingOscDiv8`.
- version 2.0.11
 - Add clearing pending errors in `TRNG_Init()`.
- version 2.0.10
 - Fixed doxygen issues.
- version 2.0.9
 - Fix HIS_CCM metrics issues.
- version 2.0.8
 - For K32L2A41A_SERIES set `TRNG_USER_CONFIG_DEFAULT_OSC_DIV` to `kTRNG_RingOscDiv4`.
- version 2.0.7
 - Fix MISRA 2004 issue rule 12.5.
- version 2.0.6
 - For KW35Z4_SERIES set `TRNG_USER_CONFIG_DEFAULT_OSC_DIV` to `kTRNG_RingOscDiv8`.
- version 2.0.5
 - Add possibility to define default TRNG configuration by device specific preprocessor macros for `FRQMIN`, `FRQMAX` and `OSCDIV`.
- version 2.0.4
 - Fix MISRA-2012 issues.
- Version 2.0.3
 - update `TRNG_Init` to restart entropy generation
- Version 2.0.2
 - fix MISRA issues
- Version 2.0.1
 - add support for KL8x and KL28Z
 - update default `OSCDIV` for K81 to divide by 2

37.6 Enumeration Type Documentation

37.6.1 enum trng_sample_mode_t

Used by [trng_config_t](#).

Enumerator

kTRNG_SampleModeVonNeumann Use von Neumann data in both Entropy shifter and Statistical Checker.

kTRNG_SampleModeRaw Use raw data into both Entropy shifter and Statistical Checker.

kTRNG_SampleModeVonNeumannRaw Use von Neumann data in Entropy shifter. Use raw data into Statistical Checker.

37.6.2 enum trng_clock_mode_t

Used by [trng_config_t](#).

Enumerator

kTRNG_ClockModeRingOscillator Ring oscillator is used to operate the TRNG (default).

kTRNG_ClockModeSystem System clock is used to operate the TRNG. This is for test use only, and indeterminate results may occur.

37.6.3 enum trng_ring_osc_div_t

Used by [trng_config_t](#).

Enumerator

kTRNG_RingOscDiv0 Ring oscillator with no divide.

kTRNG_RingOscDiv2 Ring oscillator divided-by-2.

kTRNG_RingOscDiv4 Ring oscillator divided-by-4.

kTRNG_RingOscDiv8 Ring oscillator divided-by-8.

37.7 Function Documentation

37.7.1 status_t TRNG_GetDefaultConfig ([trng_config_t](#) * *userConfig*)

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

```
* userConfig->lock = 0;
* userConfig->clockMode = kTRNG\_ClockModeRingOscillator;
* userConfig->ringOscDiv = kTRNG\_RingOscDiv0; Or to other kTRNG_RingOscDiv[2|8]
  depending on the platform.
* userConfig->sampleMode = kTRNG\_SampleModeRaw;
* userConfig->entropyDelay = 3200;
```

```

*   userConfig->sampleSize = 2500;
*   userConfig->sparseBitLimit = TRNG_USER_CONFIG_DEFAULT_SPARSE_BIT_LIMIT;
*   userConfig->retryCount = 63;
*   userConfig->longRunMaxLimit = 34;
*   userConfig->monobitLimit.maximum = 1384;
*   userConfig->monobitLimit.minimum = 1116;
*   userConfig->runBit1Limit.maximum = 405;
*   userConfig->runBit1Limit.minimum = 227;
*   userConfig->runBit2Limit.maximum = 220;
*   userConfig->runBit2Limit.minimum = 98;
*   userConfig->runBit3Limit.maximum = 125;
*   userConfig->runBit3Limit.minimum = 37;
*   userConfig->runBit4Limit.maximum = 75;
*   userConfig->runBit4Limit.minimum = 11;
*   userConfig->runBit5Limit.maximum = 47;
*   userConfig->runBit5Limit.minimum = 1;
*   userConfig->runBit6PlusLimit.maximum = 47;
*   userConfig->runBit6PlusLimit.minimum = 1;
*   userConfig->pokerLimit.maximum = 26912;
*   userConfig->pokerLimit.minimum = 24445;
*   userConfig->frequencyCountLimit.maximum = 25600;
*   userConfig->frequencyCountLimit.minimum = 1600;
*

```

Parameters

<i>userConfig</i>	User configuration structure.
-------------------	-------------------------------

Returns

If successful, returns the kStatus_TRNG_Success. Otherwise, it returns an error.

37.7.2 status_t TRNG_Init (TRNG_Type * *base*, const trng_config_t * *userConfig*)

This function initializes the TRNG. When called, the TRNG entropy generation starts immediately.

Parameters

<i>base</i>	TRNG base address
<i>userConfig</i>	Pointer to the initialization configuration structure.

Returns

If successful, returns the kStatus_TRNG_Success. Otherwise, it returns an error.

37.7.3 void TRNG_Deinit (TRNG_Type * *base*)

This function shuts down the TRNG.

Parameters

<i>base</i>	TRNG base address.
-------------	--------------------

37.7.4 status_t TRNG_GetRandomData (**TRNG_Type * base, void * data, size_t dataSize**)

This function gets random data from the TRNG.

Parameters

<i>base</i>	TRNG base address.
<i>data</i>	Pointer address used to store random data.
<i>dataSize</i>	Size of the buffer pointed by the data parameter.

Returns

random data

Chapter 38

WDOG: Watchdog Timer Driver

38.1 Overview

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

38.2 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`)

- `void WDOG_Deinit (WDOG_Type *base)`
Initializes the WDOG.
- `static void WDOG_Disable (WDOG_Type *base)`
Shuts down the WDOG.
- `static void WDOG_Disable (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.

38.3 Data Structure Documentation

38.3.1 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

38.3.2 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`

- `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`

38.4 Enumeration Type Documentation

38.4.1 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.

38.4.2 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

38.5 Function Documentation

38.5.1 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 = flase;
* wdogConfig->timeoutValue = 0xFFU;
* wdogConfig->interruptTimeValue = 0x04u;
*
```

Parameters

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

See Also

[wdog_config_t](#)

38.5.2 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

38.5.3 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).

38.5.4 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
-------------	------------------------------

38.5.5 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
-------------	------------------------------

38.5.6 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
-------------	------------------------------

38.5.7 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
-------------	------------------------------

**38.5.8 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

38.5.9 `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.

38.5.10 `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

38.5.11 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.

38.5.12 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.

38.5.13 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
-------------	------------------------------

38.5.14 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 39

XBARA: Inter-Peripheral Crossbar Switch

39.1 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.

39.2 Function

39.2.1 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.

39.2.2 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.

39.3 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_RequestInterruptEnalbe` = 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.

39.4 Data Structure Documentation

39.4.1 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`

- **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**

39.5 Enumeration Type Documentation**39.5.1 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.

39.5.2 enum xbara_request_t

Enumerator

kXBARA_RequestDisable Interrupt and DMA are disabled.

kXBARA_RequestDMAEnable DMA enabled, interrupt disabled.

kXBARA_RequestInterruptEnalbe Interrupt enabled, DMA disabled.

39.5.3 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.

39.6 Function Documentation

39.6.1 void XBARA_Init (XBARA_Type * *base*)

This function un-gates the XBARA clock.

Parameters

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

39.6.2 void XBARA_Deinit (XBARA_Type * *base*)

This function disables XBARA clock.

Parameters

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

39.6.3 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.

39.6.4 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.

39.6.5 void XBARA_ClearStatusFlags (XBARA_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	XBARA peripheral address.
<i>mask</i>	the status flags to clear.

39.6.6 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_RequestInterruptEnable;
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.

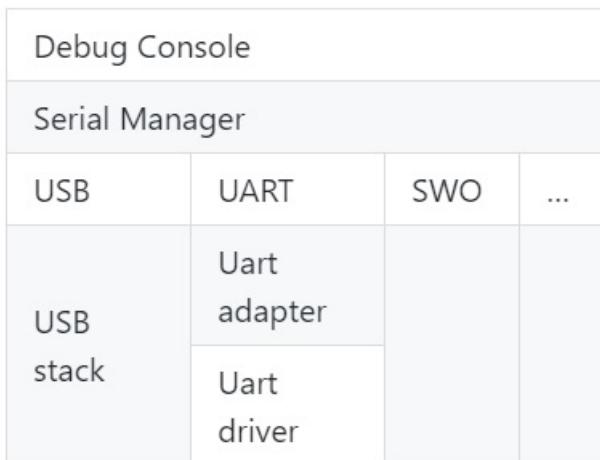
Chapter 40

Debug Console

40.1 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

40.2 Function groups

40.2.1 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);
```

40.2.2 Advanced Feature

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

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

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

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

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

length	Description
Do not support	

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

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

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

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

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

specifier	Qualifying Input	Type of argument
c	Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end.	char *
i	Integer: : Number optionally preceded with a + or - sign	int *
d	Decimal integer: Number optionally preceded with a + or - sign	int *
a, A, e, E, f, F, g, G	Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4	float *
o	Octal Integer:	int *
s	String of characters. This reads subsequent characters until a white space is found (white space characters are considered to be blank, newline, and tab).	char *
u	Unsigned decimal integer.	unsigned int *

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

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

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

```
#if SDK_DEBUGCONSOLE == DEBUGCONSOLE_DISABLE /* Disable debug console */
#define PRINTF
#define SCANF
#define PUTCHAR
#define GETCHAR
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_SDK /* Select printf, scanf, putchar, getchar of SDK
```

```

version. */
#define PRINTF DbgConsole_Printf
#define SCANF DbgConsole_Scanf
#define PUTCHAR DbgConsole_Putchar
#define GETCHAR DbgConsole_Getchar
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN /* Select printf, scanf, putchar, getchar of
toolchain. */
#define PRINTF printf
#define SCANF scanf
#define PUTCHAR putchar
#define GETCHAR getchar
#endif /* SDK_DEBUGCONSOLE */

```

40.2.3 SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART

There are two macros `SDK_DEBUGCONSOLE` and `SDK_DEBUGCONSOLE_UART` added to configure `PRINTF` and low level output peripheral.

- The macro `SDK_DEBUGCONSOLE` is used for frontend. Whether debug console redirect to toolchain or SDK or disabled, it decides which is the frontend of the debug console, Tool chain or SDK. The function can be set by the macro `SDK_DEBUGCONSOLE`.
- The macro `SDK_DEBUGCONSOLE_UART` is used for backend. It is used to decide whether provide low level IO implementation to toolchain printf and scanf. For example, within MCUXpresso, if the macro `SDK_DEBUGCONSOLE_UART` is defined, `_sys_write` and `_sys_read` will be used when `_REDLIB_` is defined; `_write` and `_read` will be used in other cases. The macro does not specifically refer to the peripheral "UART". It refers to the external peripheral similar to UART, like as USB CDC, UART, SWO, etc. So if the macro `SDK_DEBUGCONSOLE_UART` is not defined when tool-chain printf is calling, the semihosting will be used.

The following matrix show the effects of `SDK_DEBUGCONSOLE` and `SDK_DEBUGCONSOLE_UART` on `PRINTF` and `printf`. The green mark is the default setting of the debug console.

<code>SDK_DEBUGCONSOLE</code>	<code>SDK_DEBUGCONSOLE_UART</code>	<code>PRINTF</code>	<code>printf</code>
<code>DEBUGCONSOLE_- REDIRECT_TO_SDK</code>	defined	Low level peripheral*	Low level peripheral
<code>DEBUGCONSOLE_- REDIRECT_TO_SDK</code>	undefined	Low level peripheral*	semihost
<code>DEBUGCONSOLE_- REDIRECT_TO_TO- OLCHAIN</code>	defined	Low level peripheral*	Low level peripheral
<code>DEBUGCONSOLE_- REDIRECT_TO_TO- OLCHAIN</code>	undefined	semihost	semihost
<code>DEBUGCONSOLE_- DISABLE</code>	defined	No output	Low level peripheral
<code>DEBUGCONSOLE_- DISABLE</code>	undefined	No output	semihost

- * the **low level peripheral** could be USB CDC, UART, or SWO, and so on.

40.3 Typical use case

Some examples use the PUTCHAR & GETCHAR function

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

Some examples use the PRINTF function

Statement prints the string format.

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

Statement prints the hexadecimal format/

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

Statement prints the decimal floating point and unsigned decimal.

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

Some examples use the SCANF function

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

Print out failure messages using MCUXpresso SDK __assert_func:

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

Note:

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

Modules

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

Macros

- `#define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U`
Definition select redirect toolchain printf, scanf to uart or not.
- `#define DEBUGCONSOLE_REDIRECT_TO_SDK 1U`
Select SDK version printf, scanf.
- `#define DEBUGCONSOLE_DISABLE 2U`
Disable debugconsole function.
- `#define SDK_DEBUGCONSOLE DEBUGCONSOLE_REDIRECT_TO_SDK`
Definition to select sdk or toolchain printf, scanf.
- `#define PRINTF_DbgConsole_Printf`
Definition to select redirect toolchain printf, scanf to uart or not.

Typedefs

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

Functions

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

Variables

- `serial_handle_t g_serialHandle`
serial manager handle

Initialization

- `status_t DbgConsole_Init (uint8_t instance, uint32_t baudRate, serial_port_type_t device, uint32_t clkSrcFreq)`
Initializes the peripheral used for debug messages.
- `status_t DbgConsole_Deinit (void)`
De-initializes the peripheral used for debug messages.
- `status_t DbgConsole_EnterLowpower (void)`
Prepares to enter low power consumption.
- `status_t DbgConsole_ExitLowpower (void)`
Restores from low power consumption.
- `int DbgConsole_Printf (const char *fmt_s,...)`
Writes formatted output to the standard output stream.
- `int DbgConsole_Vprintf (const char *fmt_s, va_list formatStringArg)`
Writes formatted output to the standard output stream.
- `int DbgConsole_Putchar (int ch)`

- 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.

40.4 Macro Definition Documentation

40.4.1 #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U

Select toolchain printf and scanf.

40.4.2 #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U

40.4.3 #define DEBUGCONSOLE_DISABLE 2U

40.4.4 #define SDK_DEBUGCONSOLE DEBUGCONSOLE_REDIRECT_TO_SDK

The macro only support to be redefined in project setting.

40.4.5 #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.

40.5 Function Documentation

40.5.1 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
------------------------	------------------------

40.5.2 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.

40.5.3 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.

40.5.4 status_t DbgConsole_ExitLowpower (void)

This function is used to restore from low power consumption.

Returns

Indicates whether de-initialization was successful or not.

40.5.5 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.

40.5.6 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.

40.5.7 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.

40.5.8 int DbgConsole_Scanf (char * *fmt_s*, ...)

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

Note

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

Parameters

<i>fmt_s</i>	Format control string.
--------------	------------------------

Returns

Returns the number of fields successfully converted and assigned.

40.5.9 int DbgConsole_Getchar (void)

Call this function to read a character from standard input.

Note

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

Returns

Returns the character read.

40.5.10 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.

40.5.11 int DbgConsole_BlockingVprintf (const char * *fmt_s*, va_list *formatStringArg*)

Call this function to write a formatted output to the standard output stream with the blocking mode. The function will send data with blocking mode no matter the DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set or not. The function could be used in system ISR mode with DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set.

Parameters

<i>fmt_s</i>	Format control string.
<i>formatString-Arg</i>	Format arguments.

Returns

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

40.5.12 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.

40.5.13 int StrFormatPrintf (const char * *fmt*, va_list *ap*, char * *buf*, printfCb *cb*)

Note

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

Parameters

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

Returns

Number of characters to be print

40.5.14 int StrFormatScanf (const char * *line_ptr*, char * *format*, va_list *args_ptr*)

Parameters

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

Returns

Number of input items converted and assigned.

Return values

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

40.6 Semihosting

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

40.6.1 Guide Semihosting for IAR

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

Step 1: Setting up the environment

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

Step 2: Building the project

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

Step 3: Starting semihosting

1. Choose "Semihosting_IAR" project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via semihosting. Please Make sure the `SDK_DEBUGCONSOLE_UART` is not defined in project settings.
4. Start the project by choosing Project>Download and Debug.
5. Choose View>Terminal I/O to display the output from the I/O operations.

40.6.2 Guide Semihosting for Keil µVision

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

40.6.3 Guide Semihosting for MCUXpresso IDE

Step 1: Setting up the environment

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

Step 2: Building the project

1. Compile and link the project.

Step 3: Starting semihosting

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

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

40.6.4 Guide Semihosting for ARMGCC

Step 1: Setting up the environment

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

Add to "CMakeLists.txt"

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

Step 2: Building the project

1. Change "CMakeLists.txt":

```
Change "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} -specs=nano.specs")"
to "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} -specs=rdimon.specs")"
```

Replace paragraph

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-common")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffunction-sections")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fdata-sections")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffreestanding")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-builtin")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mthumb")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mapcs")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --gc-sections")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -static")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -z")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} muldefs")
```

To

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

Remove

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

2. Run "build_debug.bat" to build project

Step 3: Starting semihosting

1. Download the image and set as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\twrk64f120m\driver_examples\semihosting\armgcc\debug
d:
C:\PROGRA~2\GNUTOO~1\4BD65~1.920\bin\arm-none-eabi-gdb.exe
target remote localhost:2331
monitor reset
monitor semihosting enable
monitor semihosting thumbSWI 0xAB
monitor semihosting IOClient 1
monitor flash device = MK64FN1M0xxxx12
load semihosting_ARMGCC.elf
monitor reg pc = (0x00000004)
monitor reg sp = (0x00000000)
continue
```

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

40.7 SWO

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

40.7.1 Guide SWO for SDK

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

Step 1: Setting up the environment

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

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

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

Step 2: Building the project

Step 3: Download and run project

40.7.1.1 Guide SWO for IAR

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

Step 1: Setting up the environment

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

NOTE: Case a or c only apply at example which enable swo function, the SDK_DEBUGCONSOLE_UART definition in fsl_debug_console.h. For case a and c, Do and not do the above third step will be not affect function.

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

Step 2: Building the project

Step 3: Starting swo

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

40.7.2 Guide SWO for Keil µVision

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

Step 1: Setting up the environment

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

NOTE: Case a or c only apply at example which enable swo function, the SDK_DEBUGCONSOLE_UART definition in fsl_debug_console.h.

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

Step 3: Building the project

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

Step 4: Run the project

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

40.7.3 Guide SWO for MCUXpresso IDE

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

40.7.4 Guide SWO for ARMGCC

NOTE: ARMGCC has no library support SWO.

Chapter 41

Notification Framework

41.1 Overview

This section describes the programming interface of the Notifier driver.

41.2 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.

41.3 Data Structure Documentation

41.3.1 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**

41.3.2 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**

41.3.3 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`

41.4 Typedef Documentation

41.4.1 `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.

41.4.2 `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.

41.4.3 **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.

41.5 Enumeration Type Documentation

41.5.1 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.

41.5.2 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.

41.5.3 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.

41.5.4 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.

41.6 Function Documentation

41.6.1 **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.

41.6.2 **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.

41.6.3 **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 42

Shell

42.1 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.

42.2 Function groups

42.2.1 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>");
```

42.2.2 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.

42.2.3 Shell Operation

```
SHELL_Init(s_shellHandle, s_serialHandle, "Test@SHELL>");  
SHELL_Task(s_shellHandle);
```

Data Structures

- struct `shell_command_t`
User command data configuration structure. More...

Macros

- #define `SHELL_NON_BLOCKING_MODE` SERIAL_MANAGER_NON_BLOCKING_MODE
Whether use non-blocking mode.
- #define `SHELL_AUTO_COMPLETE` (1U)
Macro to set on/off auto-complete feature.
- #define `SHELL_BUFFER_SIZE` (64U)
Macro to set console buffer size.
- #define `SHELL_MAX_ARGS` (8U)
Macro to set maximum arguments in command.
- #define `SHELL_HISTORY_COUNT` (3U)
Macro to set maximum count of history commands.
- #define `SHELL_IGNORE_PARAMETER_COUNT` (0xFF)
Macro to bypass arguments check.
- #define `SHELL_HANDLE_SIZE`
The handle size of the shell module.
- #define `SHELL_USE_COMMON_TASK` (0U)
Macro to determine whether use common task.
- #define `SHELL_TASK_PRIORITY` (2U)
Macro to set shell task priority.
- #define `SHELL_TASK_STACK_SIZE` (1000U)
Macro to set shell task stack size.
- #define `SHELL_HANDLE_DEFINE`(name) uint32_t name[((`SHELL_HANDLE_SIZE` + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
Defines the shell handle.
- #define `SHELL_COMMAND_DEFINE`(command, descriptor, callback, paramInt)
Defines the shell command structure.
- #define `SHELL_COMMAND`(command) &g_shellCommand##command
Gets the shell command pointer.

Typedefs

- typedef void * `shell_handle_t`
The handle of the shell module.
- typedef `shell_status_t`(* `cmd_function_t`)(`shell_handle_t` shellHandle, int32_t argc, char **argv)
User command function prototype.

Enumerations

- enum `shell_status_t` {

`kStatus_SHELL_Success` = kStatus_Success,

`kStatus_SHELL_Error` = MAKE_STATUS(kStatusGroup_SHELL, 1),

`kStatus_SHELL_OpenWriteHandleFailed` = MAKE_STATUS(kStatusGroup_SHELL, 2),

`kStatus_SHELL_OpenReadHandleFailed` = MAKE_STATUS(kStatusGroup_SHELL, 3) }

Shell status.

Shell functional operation

- `shell_status_t SHELL_Init (shell_handle_t shellHandle, serial_handle_t serialHandle, char *prompt)`
Initializes the shell module.
- `shell_status_t SHELL_RegisterCommand (shell_handle_t shellHandle, shell_command_t *shellCommand)`
Registers the shell command.
- `shell_status_t SHELL_UnregisterCommand (shell_command_t *shellCommand)`
Unregisters the shell command.
- `shell_status_t SHELL_Write (shell_handle_t shellHandle, const char *buffer, uint32_t length)`
Sends data to the shell output stream.
- `int SHELL_Printf (shell_handle_t shellHandle, const char *formatString,...)`
Writes formatted output to the shell output stream.
- `shell_status_t SHELL_WriteSynchronization (shell_handle_t shellHandle, const char *buffer, uint32_t length)`
Sends data to the shell output stream with OS synchronization.
- `int SHELL_PrintfSynchronization (shell_handle_t shellHandle, const char *formatString,...)`
Writes formatted output to the shell output stream with OS synchronization.
- `void SHELL_ChangePrompt (shell_handle_t shellHandle, char *prompt)`
Change shell prompt.
- `void SHELL_PrintPrompt (shell_handle_t shellHandle)`
Print shell prompt.
- `void SHELL_Task (shell_handle_t shellHandle)`
The task function for Shell.
- `static bool SHELL_checkRunningInIsr (void)`
Check if code is running in ISR.

42.3 Data Structure Documentation

42.3.1 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`****42.4 Macro Definition Documentation****42.4.1 `#define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE`****42.4.2 `#define SHELL_AUTO_COMPLETE (1U)`****42.4.3 `#define SHELL_BUFFER_SIZE (64U)`****42.4.4 `#define SHELL_MAX_ARGS (8U)`****42.4.5 `#define SHELL_HISTORY_COUNT (3U)`****42.4.6 `#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

42.4.7 `#define SHELL_USE_COMMON_TASK (0U)`**42.4.8 `#define SHELL_TASK_PRIORITY (2U)`****42.4.9 `#define SHELL_TASK_STACK_SIZE (1000U)`**

42.4.10 #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.
-------------	--------------------------------------

42.4.11 #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.

42.4.12 #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".
----------------	--

42.5 Typedef Documentation

42.5.1 typedef shell_status_t(* cmd_function_t)(shell_handle_t shellHandle, int32_t argc, char **argv)

42.6 Enumeration Type Documentation

42.6.1 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.

42.7 Function Documentation

42.7.1 shell_status_t SHELL_Init (shell_handle_t *shellHandle*, serial_handle_t *serialHandle*, char * *prompt*)

This function must be called before calling all other Shell functions. Call operation the Shell commands with user-defined settings. The example below shows how to set up the Shell and how to call the SHELL_Init function by passing in these parameters. This is an example.

```
* static SHELL_HANDLE_DEFINE(s_shellHandle);
* SHELL_Init((shell_handle_t)s_shellHandle,
*             (serial_handle_t)s_serialHandle, "Test@SHELL>");
*
```

Parameters

<i>shellHandle</i>	Pointer to point to a memory space of size SHELL_HANDLE_SIZE allocated by the caller. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SHELL_HANDLE_DEFINE(shellHandle) ; or <code>uint32_t shellHandle[((SHELL_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>
<i>serialHandle</i>	The serial manager module handle pointer.
<i>prompt</i>	The string prompt pointer of Shell. Only the global variable can be passed.

Return values

<i>kStatus_SHELL_Success</i>	The shell initialization succeed.
<i>kStatus_SHELL_Error</i>	An error occurred when the shell is initialized.
<i>kStatus_SHELL_OpenWriteHandleFailed</i>	Open the write handle failed.
<i>kStatus_SHELL_OpenReadHandleFailed</i>	Open the read handle failed.

42.7.2 **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.

42.7.3 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.
------------------------------	--------------------------------------

42.7.4 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.

42.7.5 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.

42.7.6 **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.

42.7.7 **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.

42.7.8 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.

42.7.9 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.

42.7.10 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.
--------------------	----------------------------------

42.7.11 static bool SHELL_checkRunningInIsr(void) [inline], [static]

This function is used to check if code running in ISR.

Return values

<i>TRUE</i>	if code runing in ISR.
-------------	------------------------

Chapter 43

CODEC Driver

43.1 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](#)

43.2 CODEC Common Driver

43.2.1 Overview

The codec common driver provides a codec control abstraction interface.

Modules

- [CODEC Adapter](#)
- [CS42888 Adapter](#)
- [DA7212 Adapter](#)
- [SGTL5000 Adapter](#)
- [WM8904 Adapter](#)
- [WM8960 Adapter](#)

Data Structures

- struct [codec_config_t](#)
Initialize structure of the codec. [More...](#)
- struct [codec_capability_t](#)
codec capability [More...](#)
- struct [codec_handle_t](#)
Codec handle definition. [More...](#)

Macros

- #define [CODEC_VOLUME_MAX_VALUE](#) (100U)
codec maximum volume range

Enumerations

- enum {
 [kStatus_CODEC_NotSupport](#) = MAKE_STATUS(kStatusGroup_CODEC, 0U),
 [kStatus_CODEC_DeviceNotRegistered](#) = MAKE_STATUS(kStatusGroup_CODEC, 1U),
 [kStatus_CODEC_I2CBusInitialFailed](#),
 [kStatus_CODEC_I2CCommandTransferFailed](#) }

CODEC status.
- enum [codec_audio_protocol_t](#) {
 [kCODEC_BusI2S](#) = 0U,
 [kCODEC_BusLeftJustified](#) = 1U,
 [kCODEC_BusRightJustified](#) = 2U,
 [kCODEC_BusPCMA](#) = 3U,
 [kCODEC_BusPCMB](#) = 4U,
 [kCODEC_BusTDM](#) = 5U }

AUDIO format definition.

- enum {

kCODEC_AudioSampleRate8KHz = 8000U,
 kCODEC_AudioSampleRate11025Hz = 11025U,
 kCODEC_AudioSampleRate12KHz = 12000U,
 kCODEC_AudioSampleRate16KHz = 16000U,
 kCODEC_AudioSampleRate22050Hz = 22050U,
 kCODEC_AudioSampleRate24KHz = 24000U,
 kCODEC_AudioSampleRate32KHz = 32000U,
 kCODEC_AudioSampleRate44100Hz = 44100U,
 kCODEC_AudioSampleRate48KHz = 48000U,
 kCODEC_AudioSampleRate96KHz = 96000U,
 kCODEC_AudioSampleRate192KHz = 192000U,
 kCODEC_AudioSampleRate384KHz = 384000U }

audio sample rate definition

- enum {

kCODEC_AudioBitWidth16bit = 16U,
 kCODEC_AudioBitWidth20bit = 20U,
 kCODEC_AudioBitWidth24bit = 24U,
 kCODEC_AudioBitWidth32bit = 32U }

audio bit width

- enum codec_module_t {

kCODEC_ModuleADC = 0U,
 kCODEC_ModuleDAC = 1U,
 kCODEC_ModulePGA = 2U,
 kCODEC_ModuleHeadphone = 3U,
 kCODEC_ModuleSpeaker = 4U,
 kCODEC_ModuleLinein = 5U,
 kCODEC_ModuleLineout = 6U,
 kCODEC_ModuleVref = 7U,
 kCODEC_ModuleMicbias = 8U,
 kCODEC_ModuleMic = 9U,
 kCODEC_ModuleI2SIn = 10U,
 kCODEC_ModuleI2SOut = 11U,
 kCODEC_ModuleMixer = 12U }

audio codec module

- enum codec_module_ctrl_cmd_t { kCODEC_ModuleSwitchI2SInInterface = 0U }

audio codec module control cmd

- enum {

kCODEC_ModuleI2SInInterfacePCM = 0U,
 kCODEC_ModuleI2SInInterfaceDSD = 1U }

audio codec module digital interface

- enum {

- ```

kCODEC_RecordSourceDifferentialLine = 1U,
kCODEC_RecordSourceLineInput = 2U,
kCODEC_RecordSourceDifferentialMic = 4U,
kCODEC_RecordSourceDigitalMic = 8U,
kCODEC_RecordSourceSingleEndMic = 16U }
 audio codec module record source value
```
- enum {
 

```

kCODEC_RecordChannelLeft1 = 1U,
kCODEC_RecordChannelLeft2 = 2U,
kCODEC_RecordChannelLeft3 = 4U,
kCODEC_RecordChannelRight1 = 1U,
kCODEC_RecordChannelRight2 = 2U,
kCODEC_RecordChannelRight3 = 4U,
kCODEC_RecordChannelDifferentialPositive1 = 1U,
kCODEC_RecordChannelDifferentialPositive2 = 2U,
kCODEC_RecordChannelDifferentialPositive3 = 4U,
kCODEC_RecordChannelDifferentialNegative1 = 8U,
kCODEC_RecordChannelDifferentialNegative2 = 16U,
kCODEC_RecordChannelDifferentialNegative3 = 32U }
```

*audio codec record channel*
- enum {
 

```

kCODEC_PlaySourcePGA = 1U,
kCODEC_PlaySourceInput = 2U,
kCODEC_PlaySourceDAC = 4U,
kCODEC_PlaySourceMixerIn = 1U,
kCODEC_PlaySourceMixerInLeft = 2U,
kCODEC_PlaySourceMixerInRight = 4U,
kCODEC_PlaySourceAux = 8U }
```

*audio codec module play source value*
- enum {
 

```

kCODEC_PlayChannelHeadphoneLeft = 1U,
kCODEC_PlayChannelHeadphoneRight = 2U,
kCODEC_PlayChannelSpeakerLeft = 4U,
kCODEC_PlayChannelSpeakerRight = 8U,
kCODEC_PlayChannelLineOutLeft = 16U,
kCODEC_PlayChannelLineOutRight = 32U,
kCODEC_PlayChannelLeft0 = 1U,
kCODEC_PlayChannelRight0 = 2U,
kCODEC_PlayChannelLeft1 = 4U,
kCODEC_PlayChannelRight1 = 8U,
kCODEC_PlayChannelLeft2 = 16U,
kCODEC_PlayChannelRight2 = 32U,
kCODEC_PlayChannelLeft3 = 64U,
kCODEC_PlayChannelRight3 = 128U }
```

*codec play channel*
- enum {

```
kCODEC_VolumeHeadphoneLeft = 1U,
kCODEC_VolumeHeadphoneRight = 2U,
kCODEC_VolumeSpeakerLeft = 4U,
kCODEC_VolumeSpeakerRight = 8U,
kCODEC_VolumeLineOutLeft = 16U,
kCODEC_VolumeLineOutRight = 32U,
kCODEC_VolumeLeft0 = 1UL << 0U,
kCODEC_VolumeRight0 = 1UL << 1U,
kCODEC_VolumeLeft1 = 1UL << 2U,
kCODEC_VolumeRight1 = 1UL << 3U,
kCODEC_VolumeLeft2 = 1UL << 4U,
kCODEC_VolumeRight2 = 1UL << 5U,
kCODEC_VolumeLeft3 = 1UL << 6U,
kCODEC_VolumeRight3 = 1UL << 7U,
kCODEC_VolumeDAC = 1UL << 8U }
```

*codec volume setting*

- enum {

```

kCODEC_SupportModuleADC = 1U << 0U,
kCODEC_SupportModuleDAC = 1U << 1U,
kCODEC_SupportModulePGA = 1U << 2U,
kCODEC_SupportModuleHeadphone = 1U << 3U,
kCODEC_SupportModuleSpeaker = 1U << 4U,
kCODEC_SupportModuleLinein = 1U << 5U,
kCODEC_SupportModuleLineout = 1U << 6U,
kCODEC_SupportModuleVref = 1U << 7U,
kCODEC_SupportModuleMicbias = 1U << 8U,
kCODEC_SupportModuleMic = 1U << 9U,
kCODEC_SupportModuleI2SIn = 1U << 10U,
kCODEC_SupportModuleI2SOut = 1U << 11U,
kCODEC_SupportModuleMixer = 1U << 12U,
kCODEC_SupportModuleI2SInSwitchInterface = 1U << 13U,
kCODEC_SupportPlayChannelLeft0 = 1U << 0U,
kCODEC_SupportPlayChannelRight0 = 1U << 1U,
kCODEC_SupportPlayChannelLeft1 = 1U << 2U,
kCODEC_SupportPlayChannelRight1 = 1U << 3U,
kCODEC_SupportPlayChannelLeft2 = 1U << 4U,
kCODEC_SupportPlayChannelRight2 = 1U << 5U,
kCODEC_SupportPlayChannelLeft3 = 1U << 6U,
kCODEC_SupportPlayChannelRight3 = 1U << 7U,
kCODEC_SupportPlaySourcePGA = 1U << 8U,
kCODEC_SupportPlaySourceInput = 1U << 9U,
kCODEC_SupportPlaySourceDAC = 1U << 10U,
kCODEC_SupportPlaySourceMixerIn = 1U << 11U,
kCODEC_SupportPlaySourceMixerInLeft = 1U << 12U,
kCODEC_SupportPlaySourceMixerInRight = 1U << 13U,
kCODEC_SupportPlaySourceAux = 1U << 14U,
kCODEC_SupportRecordSourceDifferentialLine = 1U << 0U,
kCODEC_SupportRecordSourceLineInput = 1U << 1U,
kCODEC_SupportRecordSourceDifferentialMic = 1U << 2U,
kCODEC_SupportRecordSourceDigitalMic = 1U << 3U,
kCODEC_SupportRecordSourceSingleEndMic = 1U << 4U,
kCODEC_SupportRecordChannelLeft1 = 1U << 6U,
kCODEC_SupportRecordChannelLeft2 = 1U << 7U,
kCODEC_SupportRecordChannelLeft3 = 1U << 8U,
kCODEC_SupportRecordChannelRight1 = 1U << 9U,
kCODEC_SupportRecordChannelRight2 = 1U << 10U,
kCODEC_SupportRecordChannelRight3 = 1U << 11U }

```

*audio codec capability*

## Functions

- `status_t CODEC_Init (codec_handle_t *handle, codec_config_t *config)`  
*Codec initialization.*
- `status_t CODEC_Deinit (codec_handle_t *handle)`  
*Codec de-initilization.*
- `status_t CODEC_SetFormat (codec_handle_t *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `status_t CODEC_ModuleControl (codec_handle_t *handle, codec_module_ctrl_cmd_t cmd, uint32_t data)`  
*codec module control.*
- `status_t CODEC_SetVolume (codec_handle_t *handle, uint32_t channel, uint32_t volume)`  
*set audio codec pl volume.*
- `status_t CODEC_SetMute (codec_handle_t *handle, uint32_t channel, bool mute)`  
*set audio codec module mute.*
- `status_t CODEC_SetPower (codec_handle_t *handle, codec_module_t module, bool powerOn)`  
*set audio codec power.*
- `status_t CODEC_SetRecord (codec_handle_t *handle, uint32_t recordSource)`  
*codec set record source.*
- `status_t CODEC_SetRecordChannel (codec_handle_t *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)`  
*codec set record channel.*
- `status_t CODEC_SetPlay (codec_handle_t *handle, uint32_t playSource)`  
*codec set play source.*

## Driver version

- `#define FSL_CODEC_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))`  
*CLOCK driver version 2.3.0.*

### 43.2.2 Data Structure Documentation

#### 43.2.2.1 struct codec\_config\_t

##### Data Fields

- `uint32_t codecDevType`  
*codec type*
- `void *codecDevConfig`  
*Codec device specific configuration.*

### 43.2.2.2 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*

### 43.2.2.3 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*

### 43.2.3 Macro Definition Documentation

#### 43.2.3.1 #define FSL\_CODEC\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 0))

### 43.2.4 Enumeration Type Documentation

#### 43.2.4.1 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.

#### 43.2.4.2 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.

#### 43.2.4.3 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.

#### 43.2.4.4 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

#### 43.2.4.5 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

#### 43.2.4.6 enum codec\_module\_ctrl\_cmd\_t

Enumerator

*kCODEC\_ModuleSwitchI2SInInterface* module digital interface siwtch.

#### 43.2.4.7 anonymous enum

Enumerator

*kCODEC\_ModuleI2SInInterfacePCM* Pcm interface.  
*kCODEC\_ModuleI2SInInterfaceDSD* DSD interface.

#### 43.2.4.8 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

#### 43.2.4.9 anonymous enum

Enumerator

*kCODEC\_RecordChannelLeft1* left record channel 1  
*kCODEC\_RecordChannelLeft2* left record channel 2  
*kCODEC\_RecordChannelLeft3* left record channel 3  
*kCODEC\_RecordChannelRight1* right record channel 1  
*kCODEC\_RecordChannelRight2* right record channel 2  
*kCODEC\_RecordChannelRight3* right record channel 3  
*kCODEC\_RecordChannelDifferentialPositive1* differential positive record channel 1

*kCODEC\_RecordChannelDifferentialPositive2* differential positive record channel 2  
*kCODEC\_RecordChannelDifferentialPositive3* differential positive record channel 3  
*kCODEC\_RecordChannelDifferentialNegative1* differential negative record channel 1  
*kCODEC\_RecordChannelDifferentialNegative2* differential negative record channel 2  
*kCODEC\_RecordChannelDifferentialNegative3* differential negative record channel 3

#### 43.2.4.10 anonymous enum

Enumerator

*kCODEC\_PlaySourcePGA* play source PGA, bypass ADC  
*kCODEC\_PlaySourceInput* play source Input3  
*kCODEC\_PlaySourceDAC* play source DAC  
*kCODEC\_PlaySourceMixerIn* play source mixer in  
*kCODEC\_PlaySourceMixerInLeft* play source mixer in left  
*kCODEC\_PlaySourceMixerInRight* play source mixer in right  
*kCODEC\_PlaySourceAux* play source mixer in AUX

#### 43.2.4.11 anonymous enum

Enumerator

*kCODEC\_PlayChannelHeadphoneLeft* play channel headphone left  
*kCODEC\_PlayChannelHeadphoneRight* play channel headphone right  
*kCODEC\_PlayChannelSpeakerLeft* play channel speaker left  
*kCODEC\_PlayChannelSpeakerRight* play channel speaker right  
*kCODEC\_PlayChannelLineOutLeft* play channel lineout left  
*kCODEC\_PlayChannelLineOutRight* play channel lineout right  
*kCODEC\_PlayChannelLeft0* play channel left0  
*kCODEC\_PlayChannelRight0* play channel right0  
*kCODEC\_PlayChannelLeft1* play channel left1  
*kCODEC\_PlayChannelRight1* play channel right1  
*kCODEC\_PlayChannelLeft2* play channel left2  
*kCODEC\_PlayChannelRight2* play channel right2  
*kCODEC\_PlayChannelLeft3* play channel left3  
*kCODEC\_PlayChannelRight3* play channel right3

#### 43.2.4.12 anonymous enum

Enumerator

*kCODEC\_VolumeHeadphoneLeft* headphone left volume  
*kCODEC\_VolumeHeadphoneRight* headphone right volume  
*kCODEC\_VolumeSpeakerLeft* speaker left volume  
*kCODEC\_VolumeSpeakerRight* speaker right volume

*kCODEC\_VolumeLineOutLeft* lineout left volume  
*kCODEC\_VolumeLineOutRight* lineout right volume  
*kCODEC\_VolumeLeft0* left0 volume  
*kCODEC\_VolumeRight0* right0 volume  
*kCODEC\_VolumeLeft1* left1 volume  
*kCODEC\_VolumeRight1* right1 volume  
*kCODEC\_VolumeLeft2* left2 volume  
*kCODEC\_VolumeRight2* right2 volume  
*kCODEC\_VolumeLeft3* left3 volume  
*kCODEC\_VolumeRight3* right3 volume  
*kCODEC\_VolumeDAC* dac volume

#### 43.2.4.13 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

### 43.2.5 Function Documentation

#### 43.2.5.1 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.

#### 43.2.5.2 status\_t CODEC\_Deinit ( ***codec\_handle\_t \* handle*** )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

#### 43.2.5.3 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.

#### 43.2.5.4 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.

#### 43.2.5.5 status\_t CODEC\_SetVolume ( *codec\_handle\_t \* handle*, *uint32\_t channel*, *uint32\_t volume* )

Parameters

|                |                                                                                                                  |
|----------------|------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>  | codec handle.                                                                                                    |
| <i>channel</i> | audio codec volume channel, can be a value or combine value of _codec_volume_-capability or _codec_play_channel. |
| <i>volume</i>  | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.                                       |

Returns

kStatus\_Success is success, else configure failed.

#### 43.2.5.6 status\_t CODEC\_SetMute ( *codec\_handle\_t \* handle*, *uint32\_t channel*, *bool mute* )

Parameters

|                |                                                                                                                  |
|----------------|------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>  | codec handle.                                                                                                    |
| <i>channel</i> | audio codec volume channel, can be a value or combine value of _codec_volume_-capability or _codec_play_channel. |
| <i>mute</i>    | true is mute, false is unmute.                                                                                   |

Returns

kStatus\_Success is success, else configure failed.

#### 43.2.5.7 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.

#### 43.2.5.8 status\_t CODEC\_SetRecord ( *codec\_handle\_t \* handle*, *uint32\_t recordSource* )

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 43.2.5.9 status\_t CODEC\_SetRecordChannel ( *codec\_handle\_t \* handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel* )

Parameters

|                            |                                                                                                                         |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                           |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel. |

Returns

kStatus\_Success is success, else configure failed.

#### 43.2.5.10 status\_t CODEC\_SetPlay ( *codec\_handle\_t \* handle, uint32\_t playSource* )

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

## 43.3 CODEC I2C Driver

### 43.3.1 Overview

The codec common driver provides a codec control abstraction interface.

## Data Structures

- struct `codec_i2c_config_t`  
*CODEC I2C configurations structure. [More...](#)*

## Macros

- #define `CODEC_I2C_MASTER_HANDLER_SIZE` HAL\_I2C\_MASTER\_HANDLE\_SIZE  
*codec i2c handler*

## Enumerations

- enum `codec_reg_addr_t` {
   
`kCODEC_RegAddr8Bit` = 1U,  
`kCODEC_RegAddr16Bit` = 2U }  
*CODEC device register address type.*
- enum `codec_reg_width_t` {
   
`kCODEC_RegWidth8Bit` = 1U,  
`kCODEC_RegWidth16Bit` = 2U,  
`kCODEC_RegWidth32Bit` = 4U }  
*CODEC device register width.*

## Functions

- `status_t CODEC_I2C_Init` (void \*handle, uint32\_t i2cInstance, uint32\_t i2cBaudrate, uint32\_t i2cSourceClockHz)  
*Codec i2c bus initialization.*
- `status_t CODEC_I2C_Deinit` (void \*handle)  
*Codec i2c de-initilization.*
- `status_t CODEC_I2C_Send` (void \*handle, uint8\_t deviceAddress, uint32\_t subAddress, uint8\_t subaddressSize, uint8\_t \*txBuff, uint8\_t txBuffSize)  
*codec i2c send function.*
- `status_t CODEC_I2C_Receive` (void \*handle, uint8\_t deviceAddress, uint32\_t subAddress, uint8\_t subaddressSize, uint8\_t \*rxBuff, uint8\_t rxBuffSize)  
*codec i2c receive function.*

### 43.3.2 Data Structure Documentation

#### 43.3.2.1 struct codec\_i2c\_config\_t

##### Data Fields

- uint32\_t `codecI2CInstance`  
*i2c bus instance*
- uint32\_t `codecI2CSourceClock`  
*i2c bus source clock frequency*

### 43.3.3 Enumeration Type Documentation

#### 43.3.3.1 enum codec\_reg\_addr\_t

Enumerator

***kCODEC\_RegAddr8Bit*** 8-bit register address.  
***kCODEC\_RegAddr16Bit*** 16-bit register address.

#### 43.3.3.2 enum codec\_reg\_width\_t

Enumerator

***kCODEC\_RegWidth8Bit*** 8-bit register width.  
***kCODEC\_RegWidth16Bit*** 16-bit register width.  
***kCODEC\_RegWidth32Bit*** 32-bit register width.

### 43.3.4 Function Documentation

#### 43.3.4.1 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.

#### 43.3.4.2 status\_t CODEC\_I2C\_Deinit ( void \* *handle* )

Parameters

|               |                    |
|---------------|--------------------|
| <i>handle</i> | i2c master handle. |
|---------------|--------------------|

Returns

kStatus\_HAL\_I2cSuccess is success, else deinitial failed.

#### 43.3.4.3 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.

#### 43.3.4.4 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.

## 43.4 CS42888 Driver

### 43.4.1 Overview

The cs42888 driver provides a codec control interface.

## Data Structures

- struct `cs42888_audio_format_t`  
*cs42888 audio format* [More...](#)
- struct `cs42888_config_t`  
*Initialize structure of CS42888.* [More...](#)
- struct `cs42888_handle_t`  
*cs42888 handler* [More...](#)

## Macros

- #define `CS42888_I2C_HANDLER_SIZE` CODEC\_I2C\_MASTER\_HANDLER\_SIZE  
*CS42888 handle size.*
- #define `CS42888_ID` 0x01U  
*Define the register address of CS42888.*
- #define `CS42888_AOUT_MAX_VOLUME_VALUE` 0xFFU  
*CS42888 volume setting range.*
- #define `CS42888_CACHEREGNUM` 28U  
*Cache register number.*
- #define `CS42888_I2C_ADDR` 0x48U  
*CS42888 I2C address.*
- #define `CS42888_I2C_BITRATE` (100000U)  
*CS42888 I2C baudrate.*

## Typedefs

- typedef void(\* `cs42888_reset` )(bool state)  
*cs42888 reset function pointer*

## Enumerations

- enum `cs42888_func_mode` {
   
`kCS42888_ModeMasterSSM` = 0x0,  
`kCS42888_ModeMasterDSM` = 0x1,  
`kCS42888_ModeMasterQSM` = 0x2,  
`kCS42888_ModeSlave` = 0x3
 }
- CS42888 support modes.*

- enum `cs42888_module_t` {
   
kCS42888\_ModuleDACPair1 = 0x2,  
 kCS42888\_ModuleDACPair2 = 0x4,  
 kCS42888\_ModuleDACPair3 = 0x8,  
 kCS42888\_ModuleDACPair4 = 0x10,  
 kCS42888\_ModuleADCPair1 = 0x20,  
 kCS42888\_ModuleADCPair2 = 0x40 }

*Modules in CS42888 board.*

- enum `cs42888_bus_t` {
   
kCS42888\_BusLeftJustified = 0x0,  
 kCS42888\_BusI2S = 0x1,  
 kCS42888\_BusRightJustified = 0x2,  
 kCS42888\_BusOL1 = 0x4,  
 kCS42888\_BusOL2 = 0x5,  
 kCS42888\_BusTDM = 0x6 }

*CS42888 supported audio bus type.*

- enum {
   
kCS42888\_AOUT1 = 1U,  
 kCS42888\_AOUT2 = 2U,  
 kCS42888\_AOUT3 = 3U,  
 kCS42888\_AOUT4 = 4U,  
 kCS42888\_AOUT5 = 5U,  
 kCS42888\_AOUT6 = 6U,  
 kCS42888\_AOUT7 = 7U,  
 kCS42888\_AOUT8 = 8U }

*CS42888 play channel.*

## Functions

- `status_t CS42888_Init (cs42888_handle_t *handle, cs42888_config_t *config)`  
*CS42888 initialize function.*
- `status_t CS42888_Deinit (cs42888_handle_t *handle)`  
*Deinit the CS42888 codec.*
- `status_t CS42888_SetProtocol (cs42888_handle_t *handle, cs42888_bus_t protocol, uint32_t bitWidth)`  
*Set the audio transfer protocol.*
- `void CS42888_SetFuncMode (cs42888_handle_t *handle, cs42888_func_mode mode)`  
*Set CS42888 to differernt working mode.*
- `status_t CS42888_SelectFunctionalMode (cs42888_handle_t *handle, cs42888_func_mode adcMode, cs42888_func_mode dacMode)`  
*Set CS42888 to differernt functional mode.*
- `status_t CS42888_SetAOUTVolume (cs42888_handle_t *handle, uint8_t channel, uint8_t volume)`  
*Set the volume of different modules in CS42888.*
- `status_t CS42888_SetAINVolume (cs42888_handle_t *handle, uint8_t channel, uint8_t volume)`  
*Set the volume of different modules in CS42888.*
- `uint8_t CS42888_GetAOUTVolume (cs42888_handle_t *handle, uint8_t channel)`

- `uint8_t CS42888_GetAINVolume (cs42888_handle_t *handle, uint8_t channel)`  
*Get the volume of different AIN channel in CS42888.*
- `status_t CS42888_SetMute (cs42888_handle_t *handle, uint8_t channelMask)`  
*Mute modules in CS42888.*
- `status_t CS42888_SetChannelMute (cs42888_handle_t *handle, uint8_t channel, bool isMute)`  
*Mute channel modules in CS42888.*
- `status_t CS42888_SetModule (cs42888_handle_t *handle, cs42888_module_t module, bool isEnabled)`  
*Enable/disable expected devices.*
- `status_t CS42888_ConfigDataFormat (cs42888_handle_t *handle, uint32_t mclk, uint32_t sample_rate, uint32_t bits)`  
*Configure the data format of audio data.*
- `status_t CS42888_WriteReg (cs42888_handle_t *handle, uint8_t reg, uint8_t val)`  
*Write register to CS42888 using I2C.*
- `status_t CS42888_ReadReg (cs42888_handle_t *handle, uint8_t reg, uint8_t *val)`  
*Read register from CS42888 using I2C.*
- `status_t CS42888_ModifyReg (cs42888_handle_t *handle, uint8_t reg, uint8_t mask, uint8_t val)`  
*Modify some bits in the register using I2C.*

## Driver version

- `#define FSL_CS42888_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))`  
*cs42888 driver version 2.1.3.*

### 43.4.2 Data Structure Documentation

#### 43.4.2.1 struct cs42888\_audio\_format\_t

##### Data Fields

- `uint32_t mclk_HZ`  
*master clock frequency*
- `uint32_t sampleRate`  
*sample rate*
- `uint32_t bitWidth`  
*bit width*

#### 43.4.2.2 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`

### 43.4.2.3 struct cs42888\_handle\_t

#### Data Fields

- `cs42888_config_t * config`  
*cs42888 config pointer*
- `uint8_t i2cHandle [CS42888_I2C_HANDLER_SIZE]`  
*i2c handle pointer*

### 43.4.3 Macro Definition Documentation

#### 43.4.3.1 #define FSL\_CS42888\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 3))

#### 43.4.3.2 #define CS42888\_ID 0x01U

#### 43.4.3.3 #define CS42888\_I2C\_ADDR 0x48U

### 43.4.4 Enumeration Type Documentation

#### 43.4.4.1 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

#### 43.4.4.2 enum cs42888\_module\_t

Enumerator

|                                |                                                |
|--------------------------------|------------------------------------------------|
| <i>kCS42888_ModuleDACPair1</i> | DAC pair1 (AOUT1 and AOUT2) module in CS42888. |
| <i>kCS42888_ModuleDACPair2</i> | DAC pair2 (AOUT3 and AOUT4) module in CS42888. |
| <i>kCS42888_ModuleDACPair3</i> | DAC pair3 (AOUT5 and AOUT6) module in CS42888. |
| <i>kCS42888_ModuleDACPair4</i> | DAC pair4 (AOUT7 and AOUT8) module in CS42888. |
| <i>kCS42888_ModuleADCPair1</i> | ADC pair1 (AIN1 and AIN2) module in CS42888.   |
| <i>kCS42888_ModuleADCPair2</i> | ADC pair2 (AIN3 and AIN4) module in CS42888.   |

#### 43.4.4.3 enum cs42888\_bus\_t

Enumerator

|                                   |                                                  |
|-----------------------------------|--------------------------------------------------|
| <i>kCS42888_BusLeftJustified</i>  | Left justified format, up to 24 bits.            |
| <i>kCS42888_BusI2S</i>            | I2S format, up to 24 bits.                       |
| <i>kCS42888_BusRightJustified</i> | Right justified, can support 16bits and 24 bits. |
| <i>kCS42888_BusOL1</i>            | One-Line #1 mode.                                |
| <i>kCS42888_BusOL2</i>            | One-Line #2 mode.                                |
| <i>kCS42888_BusTDM</i>            | TDM mode.                                        |

#### 43.4.4.4 anonymous enum

Enumerator

|                       |       |
|-----------------------|-------|
| <i>kCS42888_AOUT1</i> | aout1 |
| <i>kCS42888_AOUT2</i> | aout2 |
| <i>kCS42888_AOUT3</i> | aout3 |
| <i>kCS42888_AOUT4</i> | aout4 |
| <i>kCS42888_AOUT5</i> | aout5 |
| <i>kCS42888_AOUT6</i> | aout6 |
| <i>kCS42888_AOUT7</i> | aout7 |
| <i>kCS42888_AOUT8</i> | aout8 |

### 43.4.5 Function Documentation

#### 43.4.5.1 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. |

#### 43.4.5.2 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. |
|---------------|-----------------------------------|

#### 43.4.5.3 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                     |

#### 43.4.5.4 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. |

#### 43.4.5.5 status\_t CS42888\_SelectFunctionalMode ( cs42888\_handle\_t \* *handle*, cs42888\_func\_mode *adcMode*, cs42888\_func\_mode *dacMode* )

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>handle</i>  | CS42888 handle structure.           |
| <i>adcMode</i> | differenht working mode of CS42888. |
| <i>dacMode</i> | differenht working mode of CS42888. |

#### 43.4.5.6 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.   |

#### 43.4.5.7 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.  |

#### 43.4.5.8 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. |

#### 43.4.5.9 `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. |

#### 43.4.5.10 `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. |

#### 43.4.5.11 `status_t CS42888_SetChannelMute ( cs42888_handle_t * handle, uint8_t channel, bool isMute )`

Parameters

|                |                                  |
|----------------|----------------------------------|
| <i>handle</i>  | CS42888 handle structure.        |
| <i>channel</i> | reference _cs42888_play_channel. |
| <i>isMute</i>  | true is mute, falase is unmute.  |

#### 43.4.5.12 `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. |

#### 43.4.5.13 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).                                                       |

#### 43.4.5.14 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. |

#### 43.4.5.15 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.                |

#### 43.4.5.16 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.                                          |

## 43.4.6 CS42888 Adapter

### 43.4.6.1 Overview

The cs42888 adapter provides a codec unify control interface.

#### Macros

- `#define HAL_CODEC_CS42888_HANDLER_SIZE (CS42888_I2C_HANDLER_SIZE + 4)`  
*codec handler size*

#### Functions

- `status_t HAL_CODEC_CS42888_Init (void *handle, void *config)`  
*Codec initialization.*
- `status_t HAL_CODEC_CS42888_Deinit (void *handle)`  
*Codec de-initilization.*
- `status_t HAL_CODEC_CS42888_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `status_t HAL_CODEC_CS42888_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)`  
*set audio codec module volume.*
- `status_t HAL_CODEC_CS42888_SetMute (void *handle, uint32_t playChannel, bool isMute)`  
*set audio codec module mute.*
- `status_t HAL_CODEC_CS42888_SetPower (void *handle, uint32_t module, bool powerOn)`  
*set audio codec module power.*
- `status_t HAL_CODEC_CS42888_SetRecord (void *handle, uint32_t recordSource)`  
*codec set record source.*
- `status_t HAL_CODEC_CS42888_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)`  
*codec set record channel.*
- `status_t HAL_CODEC_CS42888_SetPlay (void *handle, uint32_t playSource)`  
*codec set play source.*
- `status_t HAL_CODEC_CS42888_ModuleControl (void *handle, uint32_t cmd, uint32_t data)`  
*codec module control.*
- `static status_t HAL_CODEC_Init (void *handle, void *config)`  
*Codec initilization.*
- `static status_t HAL_CODEC_Deinit (void *handle)`  
*Codec de-initilization.*
- `static status_t HAL_CODEC_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `static status_t HAL_CODEC_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)`  
*set audio codec module volume.*
- `static status_t HAL_CODEC_SetMute (void *handle, uint32_t playChannel, bool isMute)`  
*set audio codec module mute.*
- `static status_t HAL_CODEC_SetPower (void *handle, uint32_t module, bool powerOn)`

- static `status_t HAL_CODEC_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- static `status_t HAL_CODEC_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- static `status_t HAL_CODEC_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- static `status_t HAL_CODEC_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

#### 43.4.6.2 Function Documentation

##### 43.4.6.2.1 `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.

##### 43.4.6.2.2 `status_t HAL_CODEC_CS42888_Deinit( void * handle )`

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

##### 43.4.6.2.3 `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.

#### 43.4.6.2.4 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.

#### 43.4.6.2.5 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.

#### 43.4.6.2.6 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.

#### 43.4.6.2.7 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.

#### 43.4.6.2.8 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.

#### 43.4.6.2.9 status\_t HAL\_CODEC\_CS42888\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 43.4.6.2.10 status\_t HAL\_CODEC\_CS42888\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* )

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

#### 43.4.6.2.11 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.

#### 43.4.6.2.12 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.

#### 43.4.6.2.13 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.

#### 43.4.6.2.14 static status\_t HAL\_CODEC\_SetVolume( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

Returns

kStatus\_Success is success, else configure failed.

#### 43.4.6.2.15 static status\_t HAL\_CODEC\_SetMute( void \* *handle*, uint32\_t *playChannel*, bool *isMute* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

Returns

kStatus\_Success is success, else configure failed.

#### 43.4.6.2.16 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.

#### 43.4.6.2.17 static status\_t HAL\_CODEC\_SetRecord ( void \* *handle*, uint32\_t *recordSource* ) [inline], [static]

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 43.4.6.2.18 static status\_t HAL\_CODEC\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* ) [inline], [static]

Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

Returns

kStatus\_Success is success, else configure failed.

**43.4.6.2.19 static status\_t HAL\_CODEC\_SetPlay ( void \* *handle*, uint32\_t *playSource* ) [inline], [static]**

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

**43.4.6.2.20 static status\_t HAL\_CODEC\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* ) [inline], [static]**

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

## 43.5 DA7212 Driver

### 43.5.1 Overview

The da7212 driver provides a codec control interface.

## Data Structures

- struct `da7212_pll_config_t`  
`da7212 pll configuration` *More...*
- struct `da7212_audio_format_t`  
`da7212 audio format` *More...*
- struct `da7212_config_t`  
`DA7212 configure structure.` *More...*
- struct `da7212_handle_t`  
`da7212 codec handler` *More...*

## Macros

- `#define DA7212_I2C_HANDLER_SIZE CODEC_I2C_MASTER_HANDLER_SIZE`  
`da7212 handle size`
- `#define DA7212_ADDRESS (0x1A)`  
`DA7212 I2C address.`
- `#define DA7212_HEADPHONE_MAX_VOLUME_VALUE 0x3FU`  
`da7212 volume setting range`

## Enumerations

- enum `da7212_Input_t` {
   
`kDA7212_Input_AUX` = 0x0,  
`kDA7212_Input_MIC1_Dig`,  
`kDA7212_Input_MIC1_An`,  
`kDA7212_Input_MIC2` }  
`DA7212 input source select.`
- enum `_da7212_play_channel` {
   
`kDA7212_HeadphoneLeft` = 1U,  
`kDA7212_HeadphoneRight` = 2U,  
`kDA7212_Speaker` = 4U }  
`da7212 play channel`
- enum `da7212_Output_t` {
   
`kDA7212_Output_HP` = 0x0,  
`kDA7212_Output_SP` }  
`DA7212 output device select.`

- enum \_da7212\_module {
 kDA7212\_ModuleADC,
 kDA7212\_ModuleDAC,
 kDA7212\_ModuleHeadphone,
 kDA7212\_ModuleSpeaker }
   
*DA7212 module.*
- enum da7212\_dac\_source\_t {
 kDA7212\_DACSourceADC = 0x0U,
 kDA7212\_DACSourceInputStream = 0x3U }
   
*DA7212 functionality.*
- enum da7212\_volume\_t {
 kDA7212\_DACGainMute = 0x7,
 kDA7212\_DACGainM72DB = 0x17,
 kDA7212\_DACGainM60DB = 0x1F,
 kDA7212\_DACGainM54DB = 0x27,
 kDA7212\_DACGainM48DB = 0x2F,
 kDA7212\_DACGainM42DB = 0x37,
 kDA7212\_DACGainM36DB = 0x3F,
 kDA7212\_DACGainM30DB = 0x47,
 kDA7212\_DACGainM24DB = 0x4F,
 kDA7212\_DACGainM18DB = 0x57,
 kDA7212\_DACGainM12DB = 0x5F,
 kDA7212\_DACGainM6DB = 0x67,
 kDA7212\_DACGain0DB = 0x6F,
 kDA7212\_DACGain6DB = 0x77,
 kDA7212\_DACGain12DB = 0x7F }
   
*DA7212 volume.*
- enum da7212\_protocol\_t {
 kDA7212\_BusI2S = 0x0,
 kDA7212\_BusLeftJustified,
 kDA7212\_BusRightJustified,
 kDA7212\_BusDSPMode }
   
*The audio data transfer protocol choice.*
- enum da7212\_sys\_clk\_source\_t {
 kDA7212\_SysClkSourceMCLK = 0U,
 kDA7212\_SysClkSourcePLL = 1U << 14 }
   
*da7212 system clock source*
- enum da7212\_pll\_clk\_source\_t { kDA7212\_PLLClkSourceMCLK = 0U }
   
*DA7212 pll clock source.*
- enum da7212\_pll\_out\_clk\_t {
 kDA7212\_PLLOutputClk11289600 = 11289600U,
 kDA7212\_PLLOutputClk12288000 = 12288000U }
   
*DA7212 output clock frequency.*
- enum da7212\_master\_bits\_t { }

```

kDA7212_MasterBits32PerFrame = 0U,
kDA7212_MasterBits64PerFrame = 1U,
kDA7212_MasterBits128PerFrame = 2U,
kDA7212_MasterBits256PerFrame = 3U }
 master mode bits per frame

```

## Functions

- `status_t DA7212_Init (da7212_handle_t *handle, da7212_config_t *codecConfig)`  
*DA7212 initialize function.*
- `status_t DA7212_ConfigAudioFormat (da7212_handle_t *handle, uint32_t masterClock_Hz, uint32_t sampleRate_Hz, uint32_t dataBits)`  
*Set DA7212 audio format.*
- `status_t DA7212_SetPLLConfig (da7212_handle_t *handle, da7212_pll_config_t *config)`  
*DA7212 set PLL configuration This function will enable the GPIO1 FLL clock output function, so user can see the generated fll output clock frequency from WM8904 GPIO1.*
- `void DA7212_ChangeHPVolume (da7212_handle_t *handle, da7212_volume_t volume)`  
*Set DA7212 playback volume.*
- `void DA7212_Mute (da7212_handle_t *handle, bool isMuted)`  
*Mute or unmute DA7212.*
- `void DA7212_ChangeInput (da7212_handle_t *handle, da7212_Input_t DA7212_Input)`  
*Set the input data source of DA7212.*
- `void DA7212_ChangeOutput (da7212_handle_t *handle, da7212_Output_t DA7212_Output)`  
*Set the output device of DA7212.*
- `status_t DA7212_SetChannelVolume (da7212_handle_t *handle, uint32_t channel, uint32_t volume)`  
*Set module volume.*
- `status_t DA7212_SetChannelMute (da7212_handle_t *handle, uint32_t channel, bool isMute)`  
*Set module mute.*
- `status_t DA7212_SetProtocol (da7212_handle_t *handle, da7212_protocol_t protocol)`  
*Set protocol for DA7212.*
- `status_t DA7212_SetMasterModeBits (da7212_handle_t *handle, uint32_t bitWidth)`  
*Set master mode bits per frame for DA7212.*
- `status_t DA7212_WriteRegister (da7212_handle_t *handle, uint8_t u8Register, uint8_t u8RegisterData)`  
*Write a register for DA7212.*
- `status_t DA7212_ReadRegister (da7212_handle_t *handle, uint8_t u8Register, uint8_t *pu8RegisterData)`  
*Get a register value of DA7212.*
- `status_t DA7212_Deinit (da7212_handle_t *handle)`  
*Deinit DA7212.*

## Driver version

- `#define FSL_DA7212_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))`  
*CLOCK driver version 2.2.2.*

## 43.5.2 Data Structure Documentation

### 43.5.2.1 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*

### 43.5.2.2 struct da7212\_audio\_format\_t

#### Data Fields

- `uint32_t mclk_HZ`  
*master clock frequency*
- `uint32_t sampleRate`  
*sample rate*
- `uint32_t bitWidth`  
*bit width*
- `bool isBclkInvert`  
*bit clock invertet*

### 43.5.2.3 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`

#### 43.5.2.4 struct da7212\_handle\_t

##### Data Fields

- `da7212_config_t * config`  
*da7212 config pointer*
- `uint8_t i2cHandle [DA7212_I2C_HANDLER_SIZE]`  
*i2c handle*

#### 43.5.3 Macro Definition Documentation

##### 43.5.3.1 #define FSL\_DA7212\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 2))

#### 43.5.4 Enumeration Type Documentation

##### 43.5.4.1 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.

##### 43.5.4.2 enum \_da7212\_play\_channel

Enumerator

- kDA7212\_HeadphoneLeft* headphone left
- kDA7212\_HeadphoneRight* headphone right
- kDA7212\_Speaker* speaker channel

##### 43.5.4.3 enum da7212\_Output\_t

Enumerator

- kDA7212\_Output\_HP* Output to headphone.
- kDA7212\_Output\_SP* Output to speaker.

#### 43.5.4.4 enum \_da7212\_module

Enumerator

*kDA7212\_ModuleADC* module ADC  
*kDA7212\_ModuleDAC* module DAC  
*kDA7212\_ModuleHeadphone* module headphone  
*kDA7212\_ModuleSpeaker* module speaker

#### 43.5.4.5 enum da7212\_dac\_source\_t

Enumerator

*kDA7212\_DACSourceADC* DAC source from ADC.  
*kDA7212\_DACSourceInputStream* DAC source from.

#### 43.5.4.6 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.

#### 43.5.4.7 enum da7212\_protocol\_t

Enumerator

*kDA7212\_BusI2S* I2S Type.  
*kDA7212\_BusLeftJustified* Left justified.  
*kDA7212\_BusRightJustified* Right Justified.  
*kDA7212\_BusDSPMode* DSP mode.

#### 43.5.4.8 enum da7212\_sys\_clk\_source\_t

Enumerator

*kDA7212\_SysClkSourceMCLK* da7212 system clock soure from MCLK

*kDA7212\_SysClkSourcePLL* da7212 system clock soure from pLL

#### 43.5.4.9 enum da7212\_pll\_clk\_source\_t

Enumerator

*kDA7212\_PLLClkSourceMCLK* DA7212 PLL clock source from MCLK.

#### 43.5.4.10 enum da7212\_pll\_out\_clk\_t

Enumerator

*kDA7212\_PLLOutputClk11289600* output 112896000U

*kDA7212\_PLLOutputClk12288000* output 12288000U

#### 43.5.4.11 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

### 43.5.5 Function Documentation

#### 43.5.5.1 status\_t DA7212\_Init ( da7212\_handle\_t \* *handle*, da7212\_config\_t \* *codecConfig* )

Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | DA7212 handle pointer. |
|---------------|------------------------|

|                    |                                                                                                                                                                                                                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>codecConfig</i> | Codec configure structure. This parameter can be NULL, if NULL, set as default settings. The default setting:<br><br><pre>* sgtl_init_t codec_config * codec_config.route = kDA7212_RoutePlayback * codec_config.bus = <b>kDA7212_BusI2S</b> * codec_config.isMaster = <b>false</b> *</pre> |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

#### 43.5.5.2 status\_t DA7212\_ConfigAudioFormat ( *da7212\_handle\_t \* handle, uint32\_t masterClock\_Hz, uint32\_t sampleRate\_Hz, uint32\_t dataBits* )

Parameters

|                       |                                                                                                                                                                                                                                        |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>         | DA7212 handle pointer.                                                                                                                                                                                                                 |
| <i>masterClock_Hz</i> | Master clock frequency in Hz. If DA7212 is slave, use the frequency of master, if DA7212 as master, it should be 1228000 while sample rate frequency is 8k/12K/16-K/24K/32K/48K/96K, 11289600 whie sample rate is 11.025K/22.05K/44.1K |
| <i>sampleRate_Hz</i>  | Sample rate frequency in Hz.                                                                                                                                                                                                           |
| <i>dataBits</i>       | How many bits in a word of a audio frame, DA7212 only supports 16/20/24/32 bits.                                                                                                                                                       |

#### 43.5.5.3 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. |

#### 43.5.5.4 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. |

#### 43.5.5.5 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. |

#### 43.5.5.6 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.     |

#### 43.5.5.7 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. |

#### 43.5.5.8 status\_t DA7212\_SetChannelVolume ( da7212\_handle\_t \* *handle*, uint32\_t *channel*, uint32\_t *volume* )

Parameters

|                |                                                    |
|----------------|----------------------------------------------------|
| <i>handle</i>  | DA7212 handle pointer.                             |
| <i>channel</i> | shoule be a value of _da7212_channel.              |
| <i>volume</i>  | volume range 0 - 0x3F mapped to range -57dB - 6dB. |

#### 43.5.5.9 status\_t DA7212\_SetChannelMute ( *da7212\_handle\_t \* handle, uint32\_t channel, bool isMute* )

Parameters

|                |                                       |
|----------------|---------------------------------------|
| <i>handle</i>  | DA7212 handle pointer.                |
| <i>channel</i> | shoule be a value of _da7212_channel. |
| <i>isMute</i>  | true is mute, false is unmute.        |

#### 43.5.5.10 status\_t DA7212\_SetProtocol ( *da7212\_handle\_t \* handle, da7212\_protocol\_t protocol* )

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>handle</i>   | DA7212 handle pointer. |
| <i>protocol</i> | da7212_protocol_t.     |

#### 43.5.5.11 status\_t DA7212\_SetMasterModeBits ( *da7212\_handle\_t \* handle, uint32\_t bitWidth* )

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>handle</i>   | DA7212 handle pointer. |
| <i>bitWidth</i> | audio data bitwidth.   |

#### 43.5.5.12 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       |

#### 43.5.5.13 status\_t DA7212\_ReadRegister ( da7212\_handle\_t \* *handle*, uint8\_t *u8Register*, uint8\_t \* *pu8RegisterData* )

Parameters

|                        |                                                |
|------------------------|------------------------------------------------|
| <i>handle</i>          | DA7212 handle pointer.                         |
| <i>u8Register</i>      | DA7212 register address to be read.            |
| <i>pu8RegisterData</i> | Pointer where the read out value to be stored. |

#### 43.5.5.14 status\_t DA7212\_Deinit ( da7212\_handle\_t \* *handle* )

Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | DA7212 handle pointer. |
|---------------|------------------------|

## 43.5.6 DA7212 Adapter

### 43.5.6.1 Overview

The da7212 adapter provides a codec unify control interface.

#### Macros

- `#define HAL_CODEC_DA7212_HANDLER_SIZE (DA7212_I2C_HANDLER_SIZE + 4)`  
*codec handler size*

#### Functions

- `status_t HAL_CODEC_DA7212_Init (void *handle, void *config)`  
*Codec initialization.*
- `status_t HAL_CODEC_DA7212_Deinit (void *handle)`  
*Codec de-initilization.*
- `status_t HAL_CODEC_DA7212_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `status_t HAL_CODEC_DA7212_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)`  
*set audio codec module volume.*
- `status_t HAL_CODEC_DA7212_SetMute (void *handle, uint32_t playChannel, bool isMute)`  
*set audio codec module mute.*
- `status_t HAL_CODEC_DA7212_SetPower (void *handle, uint32_t module, bool powerOn)`  
*set audio codec module power.*
- `status_t HAL_CODEC_DA7212_SetRecord (void *handle, uint32_t recordSource)`  
*codec set record source.*
- `status_t HAL_CODEC_DA7212_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)`  
*codec set record channel.*
- `status_t HAL_CODEC_DA7212_SetPlay (void *handle, uint32_t playSource)`  
*codec set play source.*
- `status_t HAL_CODEC_DA7212_ModuleControl (void *handle, uint32_t cmd, uint32_t data)`  
*codec module control.*
- `static status_t HAL_CODEC_Init (void *handle, void *config)`  
*Codec initilization.*
- `static status_t HAL_CODEC_Deinit (void *handle)`  
*Codec de-initilization.*
- `static status_t HAL_CODEC_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `static status_t HAL_CODEC_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)`  
*set audio codec module volume.*
- `static status_t HAL_CODEC_SetMute (void *handle, uint32_t playChannel, bool isMute)`  
*set audio codec module mute.*
- `static status_t HAL_CODEC_SetPower (void *handle, uint32_t module, bool powerOn)`

- static `status_t HAL_CODEC_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- static `status_t HAL_CODEC_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- static `status_t HAL_CODEC_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- static `status_t HAL_CODEC_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

### 43.5.6.2 Function Documentation

#### 43.5.6.2.1 `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.

#### 43.5.6.2.2 `status_t HAL_CODEC_DA7212_Deinit( void * handle )`

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

#### 43.5.6.2.3 `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.

#### 43.5.6.2.4 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.

#### 43.5.6.2.5 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.

#### 43.5.6.2.6 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.

#### 43.5.6.2.7 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.

#### 43.5.6.2.8 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.

#### 43.5.6.2.9 status\_t HAL\_CODEC\_DA7212\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 43.5.6.2.10 status\_t HAL\_CODEC\_DA7212\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* )

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

#### 43.5.6.2.11 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.

#### 43.5.6.2.12 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.

#### 43.5.6.2.13 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.

#### 43.5.6.2.14 static status\_t HAL\_CODEC\_SetVolume( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

Returns

kStatus\_Success is success, else configure failed.

#### 43.5.6.2.15 static status\_t HAL\_CODEC\_SetMute( void \* *handle*, uint32\_t *playChannel*, bool *isMute* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

Returns

kStatus\_Success is success, else configure failed.

#### 43.5.6.2.16 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.

#### 43.5.6.2.17 static status\_t HAL\_CODEC\_SetRecord ( void \* *handle*, uint32\_t *recordSource* ) [inline], [static]

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 43.5.6.2.18 static status\_t HAL\_CODEC\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* ) [inline], [static]

Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

Returns

kStatus\_Success is success, else configure failed.

**43.5.6.2.19 static status\_t HAL\_CODEC\_SetPlay ( void \* *handle*, uint32\_t *playSource* ) [inline], [static]**

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

**43.5.6.2.20 static status\_t HAL\_CODEC\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* ) [inline], [static]**

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

## 43.6 SGTL5000 Driver

### 43.6.1 Overview

The sgtl5000 driver provides a codec control interface.

### Data Structures

- struct `sgtl_audio_format_t`  
*Audio format configuration. [More...](#)*
- struct `sgtl_config_t`  
*Initialize structure of sgtl5000. [More...](#)*
- struct `sgtl_handle_t`  
*SGTL codec handler. [More...](#)*

### Macros

- #define `CHIP_ID` 0x0000U  
*Define the register address of sgtl5000.*
- #define `SGTL5000_HEADPHONE_MAX_VOLUME_VALUE` 0x7FU  
*SGTL5000 volume setting range.*
- #define `SGTL5000_I2C_ADDR` 0x0A  
*SGTL5000 I2C address.*
- #define `SGTL_I2C_HANDLER_SIZE` CODEC\_I2C\_MASTER\_HANDLER\_SIZE  
*sgtl handle size*
- #define `SGTL_I2C_BITRATE` 100000U  
*sgtl i2c baudrate*

### Enumerations

- enum `sgtl_module_t` {
   
`kSGTL_ModuleADC` = 0x0,  
`kSGTL_ModuleDAC`,  
`kSGTL_ModuleDAP`,  
`kSGTL_ModuleHP`,  
`kSGTL_ModuleI2SIN`,  
`kSGTL_ModuleI2SOUT`,  
`kSGTL_ModuleLineIn`,  
`kSGTL_ModuleLineOut`,  
`kSGTL_ModuleMicin` }
- Modules in Sglt5000 board.
- enum `sgtl_route_t` {

```

kSGTL_RouteBypass = 0x0,
kSGTL_RoutePlayback,
kSGTL_RoutePlaybackandRecord,
kSGTL_RoutePlaybackwithDAP,
kSGTL_RoutePlaybackwithDAPandRecord,
kSGTL_RouteRecord }

Sgtl5000 data route.
• enum sgtl_protocol_t {
 kSGTL_BusI2S = 0x0,
 kSGTL_BusLeftJustified,
 kSGTL_BusRightJustified,
 kSGTL_BusPCMA,
 kSGTL_BusPCMB }

The audio data transfer protocol choice.
• enum {
 kSGTL_HeadphoneLeft = 0,
 kSGTL_HeadphoneRight = 1,
 kSGTL_LineoutLeft = 2,
 kSGTL_LineoutRight = 3 }

sgtl play channel
• enum {
 kSGTL_RecordSourceLineIn = 0U,
 kSGTL_RecordSourceMic = 1U }

sgtl record source _sgtl_record_source
• enum {
 kSGTL_PlaySourceLineIn = 0U,
 kSGTL_PlaySourceDAC = 1U }

sgtl play source _stgl_play_source
• enum sgtl_sclk_edge_t {
 kSGTL_SclkValidEdgeRising = 0U,
 kSGTL_SclkValidEdgeFailling = 1U }

SGTL SCLK valid edge.

```

## Functions

- `status_t SGTL_Init (sgtl_handle_t *handle, sgtl_config_t *config)`  
*sgtl5000 initialize function.*
- `status_t SGTL_SetDataRoute (sgtl_handle_t *handle, sgtl_route_t route)`  
*Set audio data route in sgtl5000.*
- `status_t SGTL_SetProtocol (sgtl_handle_t *handle, sgtl_protocol_t protocol)`  
*Set the audio transfer protocol.*
- `void SGTL_SetMasterSlave (sgtl_handle_t *handle, bool master)`  
*Set sgtl5000 as master or slave.*
- `status_t SGTL_SetVolume (sgtl_handle_t *handle, sgtl_module_t module, uint32_t volume)`  
*Set the volume of different modules in sgtl5000.*
- `uint32_t SGTL_GetVolume (sgtl_handle_t *handle, sgtl_module_t module)`  
*Get the volume of different modules in sgtl5000.*

- `status_t SGTL_SetMute (sgtl_handle_t *handle, sgtl_module_t module, bool mute)`  
*Mute/unmute modules in sgtl5000.*
- `status_t SGTL_EnableModule (sgtl_handle_t *handle, sgtl_module_t module)`  
*Enable expected devices.*
- `status_t SGTL_DisableModule (sgtl_handle_t *handle, sgtl_module_t module)`  
*Disable expected devices.*
- `status_t SGTL_Deinit (sgtl_handle_t *handle)`  
*Deinit the sgtl5000 codec.*
- `status_t SGTL_ConfigDataFormat (sgtl_handle_t *handle, uint32_t mclk, uint32_t sample_rate, uint32_t bits)`  
*Configure the data format of audio data.*
- `status_t SGTL_SetPlay (sgtl_handle_t *handle, uint32_t playSource)`  
*select SGTL codec play source.*
- `status_t SGTL_SetRecord (sgtl_handle_t *handle, uint32_t recordSource)`  
*select SGTL codec record source.*
- `status_t SGTL_WriteReg (sgtl_handle_t *handle, uint16_t reg, uint16_t val)`  
*Write register to sgtl using I2C.*
- `status_t SGTL_ReadReg (sgtl_handle_t *handle, uint16_t reg, uint16_t *val)`  
*Read register from sgtl using I2C.*
- `status_t SGTL_ModifyReg (sgtl_handle_t *handle, uint16_t reg, uint16_t clr_mask, uint16_t val)`  
*Modify some bits in the register using I2C.*

## Driver version

- `#define FSL_SGTL5000_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`  
*CLOCK driver version 2.1.1.*

### 43.6.2 Data Structure Documentation

#### 43.6.2.1 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*

#### 43.6.2.2 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.

### 43.6.2.3 struct `sgtl_handle_t`

#### Data Fields

- `sgtl_config_t * config`  
*sgtl config pointer*
- `uint8_t i2cHandle [SGTL_I2C_HANDLER_SIZE]`  
*i2c handle*

### 43.6.3 Macro Definition Documentation

#### 43.6.3.1 #define `FSL_SGTL5000_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 1)`)

#### 43.6.3.2 #define `CHIP_ID` `0x0000U`

#### 43.6.3.3 #define `SGTL5000_I2C_ADDR` `0x0A`

### 43.6.4 Enumeration Type Documentation

#### 43.6.4.1 enum `sgtl_module_t`

Enumerator

- `kSGTL_ModuleADC` ADC module in SGTL5000.
- `kSGTL_ModuleDAC` DAC module in SGTL5000.
- `kSGTL_ModuleDAP` DAP module in SGTL5000.
- `kSGTL_ModuleHP` Headphone module in SGTL5000.

*kSGTL\_ModuleI2SIN* I2S-IN module in SGTL5000.  
*kSGTL\_ModuleI2SOUT* I2S-OUT module in SGTL5000.  
*kSGTL\_ModuleLineIn* Line-in moudle in SGTL5000.  
*kSGTL\_ModuleLineOut* Line-out module in SGTL5000.  
*kSGTL\_ModuleMicin* Micphone module in SGTL5000.

#### 43.6.4.2 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.

#### 43.6.4.3 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.

#### 43.6.4.4 anonymous enum

Enumerator

*kSGTL\_HeadphoneLeft* headphone left channel  
*kSGTL\_HeadphoneRight* headphone right channel  
*kSGTL\_LineoutLeft* lineout left channel  
*kSGTL\_LineoutRight* lineout right channel

#### 43.6.4.5 anonymous enum

Enumerator

*kSGTL\_RecordSourceLineIn* record source line in  
*kSGTL\_RecordSourceMic* record source single end

#### 43.6.4.6 anonymous enum

Enumerator

*kSGTL\_PlaySourceLineIn* play source line in  
*kSGTL\_PlaySourceDAC* play source line in

#### 43.6.4.7 enum sgtl\_sclk\_edge\_t

Enumerator

*kSGTL\_SclkValidEdgeRising* SCLK valid edge.  
*kSGTL\_SclkValidEdgeFailling* SCLK failling edge.

### 43.6.5 Function Documentation

#### 43.6.5.1 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

#### 43.6.5.2 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. |

#### 43.6.5.3 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. |

#### 43.6.5.4 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. |

#### 43.6.5.5 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. |

#### 43.6.5.6 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.

#### 43.6.5.7 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.   |

#### 43.6.5.8 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. |

#### 43.6.5.9 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. |

#### 43.6.5.10 status\_t SGTL\_Deinit ( *sgtl\_handle\_t \* handle* )

Shut down Sgtl5000 modules.

Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>handle</i> | Sgtl5000 handle structure pointer. |
|---------------|------------------------------------|

#### 43.6.5.11 status\_t SGTL\_ConfigDataFormat ( *sgtl\_handle\_t \* handle*, *uint32\_t mclk*, *uint32\_t sample\_rate*, *uint32\_t bits* )

This function would configure the registers about the sample rate, bit depths.

Parameters

|                    |                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>      | Sgtl5000 handle structure pointer.                                                                                                            |
| <i>mclk</i>        | Master clock frequency of I2S.                                                                                                                |
| <i>sample_rate</i> | Sample rate of audio file running in sgtl5000. Sgtl5000 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate. |
| <i>bits</i>        | Bit depth of audio file (Sgtl5000 only supports 16bit, 20bit, 24bit and 32 bit in HW).                                                        |

#### 43.6.5.12 status\_t SGTL\_SetPlay ( *sgtl\_handle\_t \* handle, uint32\_t playSource* )

Parameters

|                   |                                                 |
|-------------------|-------------------------------------------------|
| <i>handle</i>     | Sgtl5000 handle structure pointer.              |
| <i>playSource</i> | play source value, reference _sgtl_play_source. |

Returns

kStatus\_Success, else failed.

#### 43.6.5.13 status\_t SGTL\_SetRecord ( *sgtl\_handle\_t \* handle, uint32\_t recordSource* )

Parameters

|                     |                                                     |
|---------------------|-----------------------------------------------------|
| <i>handle</i>       | Sgtl5000 handle structure pointer.                  |
| <i>recordSource</i> | record source value, reference _sgtl_record_source. |

Returns

kStatus\_Success, else failed.

#### 43.6.5.14 status\_t SGTL\_WriteReg ( *sgtl\_handle\_t \* handle, uint16\_t reg, uint16\_t val* )

Parameters

|               |                            |
|---------------|----------------------------|
| <i>handle</i> | Sgtl5000 handle structure. |
|---------------|----------------------------|

|            |                                         |
|------------|-----------------------------------------|
| <i>reg</i> | The register address in sgtl.           |
| <i>val</i> | Value needs to write into the register. |

#### 43.6.5.15 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.             |

#### 43.6.5.16 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.                                          |

## 43.6.6 SGTL5000 Adapter

### 43.6.6.1 Overview

The sgtl5000 adapter provides a codec unify control interface.

### Macros

- `#define HAL_CODEC_SGTL_HANDLER_SIZE (SGTL_I2C_HANDLER_SIZE + 4)`  
*codec handler size*

### Functions

- `status_t HAL_CODEC_SGTL5000_Init (void *handle, void *config)`  
*Codec initialization.*
- `status_t HAL_CODEC_SGTL5000_Deinit (void *handle)`  
*Codec de-initilization.*
- `status_t HAL_CODEC_SGTL5000_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `status_t HAL_CODEC_SGTL5000_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)`  
*set audio codec module volume.*
- `status_t HAL_CODEC_SGTL5000_SetMute (void *handle, uint32_t playChannel, bool isMute)`  
*set audio codec module mute.*
- `status_t HAL_CODEC_SGTL5000_SetPower (void *handle, uint32_t module, bool powerOn)`  
*set audio codec module power.*
- `status_t HAL_CODEC_SGTL5000_SetRecord (void *handle, uint32_t recordSource)`  
*codec set record source.*
- `status_t HAL_CODEC_SGTL5000_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)`  
*codec set record channel.*
- `status_t HAL_CODEC_SGTL5000_SetPlay (void *handle, uint32_t playSource)`  
*codec set play source.*
- `status_t HAL_CODEC_SGTL5000_ModuleControl (void *handle, uint32_t cmd, uint32_t data)`  
*codec module control.*
- `static status_t HAL_CODEC_Init (void *handle, void *config)`  
*Codec initilization.*
- `static status_t HAL_CODEC_Deinit (void *handle)`  
*Codec de-initilization.*
- `static status_t HAL_CODEC_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `static status_t HAL_CODEC_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)`  
*set audio codec module volume.*
- `static status_t HAL_CODEC_SetMute (void *handle, uint32_t playChannel, bool isMute)`  
*set audio codec module mute.*
- `static status_t HAL_CODEC_SetPower (void *handle, uint32_t module, bool powerOn)`

- static `status_t HAL_CODEC_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- static `status_t HAL_CODEC_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- static `status_t HAL_CODEC_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- static `status_t HAL_CODEC_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

### 43.6.6.2 Function Documentation

#### 43.6.6.2.1 `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.

#### 43.6.6.2.2 `status_t HAL_CODEC_SGTL5000_Deinit( void * handle )`

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

#### 43.6.6.2.3 `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.

#### 43.6.6.2.4 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.

#### 43.6.6.2.5 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.

#### 43.6.6.2.6 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.

#### 43.6.6.2.7 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.

#### 43.6.6.2.8 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.

#### 43.6.6.2.9 status\_t HAL\_CODEC\_SGTL5000\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 43.6.6.2.10 status\_t HAL\_CODEC\_SGTL5000\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* )

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

#### 43.6.6.2.11 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.

#### 43.6.6.2.12 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.

#### 43.6.6.2.13 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.

#### 43.6.6.2.14 static status\_t HAL\_CODEC\_SetVolume( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

Returns

kStatus\_Success is success, else configure failed.

#### 43.6.6.2.15 static status\_t HAL\_CODEC\_SetMute( void \* *handle*, uint32\_t *playChannel*, bool *isMute* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

Returns

kStatus\_Success is success, else configure failed.

#### 43.6.6.2.16 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.

#### 43.6.6.2.17 static status\_t HAL\_CODEC\_SetRecord ( void \* *handle*, uint32\_t *recordSource* ) [inline], [static]

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 43.6.6.2.18 static status\_t HAL\_CODEC\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* ) [inline], [static]

Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

Returns

kStatus\_Success is success, else configure failed.

**43.6.6.2.19 static status\_t HAL\_CODEC\_SetPlay ( void \* *handle*, uint32\_t *playSource* ) [inline], [static]**

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

**43.6.6.2.20 static status\_t HAL\_CODEC\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* ) [inline], [static]**

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

## 43.7 WM8960 Driver

### 43.7.1 Overview

The wm8960 driver provides a codec control interface.

## Data Structures

- struct `wm8960_audio_format_t`  
*wm8960 audio format More...*
- struct `wm8960_master_sysclk_config_t`  
*wm8960 master system clock configuration More...*
- struct `wm8960_config_t`  
*Initialize structure of WM8960. More...*
- struct `wm8960_handle_t`  
*wm8960 codec handler More...*

## Macros

- #define `WM8960_I2C_HANDLER_SIZE` CODEC\_I2C\_MASTER\_HANDLER\_SIZE  
*wm8960 handle size*
- #define `WM8960_LINVOL` 0x0U  
*Define the register address of WM8960.*
- #define `WM8960_CACHEREGNUM` 56U  
*Cache register number.*
- #define `WM8960_CLOCK2_BCLK_DIV_MASK` 0xFU  
*WM8960 CLOCK2 bits.*
- #define `WM8960_IFACE1_FORMAT_MASK` 0x03U  
*WM8960\_IFACE1 FORMAT bits.*
- #define `WM8960_IFACE1_WL_MASK` 0x0CU  
*WM8960\_IFACE1 WL bits.*
- #define `WM8960_IFACE1_LRP_MASK` 0x10U  
*WM8960\_IFACE1 LRP bit.*
- #define `WM8960_IFACE1_DLRSWAP_MASK` 0x20U  
*WM8960\_IFACE1 DLRSWAP bit.*
- #define `WM8960_IFACE1_MS_MASK` 0x40U  
*WM8960\_IFACE1 MS bit.*
- #define `WM8960_IFACE1_BCLKINV_MASK` 0x80U  
*WM8960\_IFACE1 BCLKINV bit.*
- #define `WM8960_IFACE1_ALRSWAP_MASK` 0x100U  
*WM8960\_IFACE1 ALRSWAP bit.*
- #define `WM8960_POWER1_VREF_MASK` 0x40U  
*WM8960\_POWER1.*
- #define `WM8960_POWER2_DACL_MASK` 0x100U  
*WM8960\_POWER2.*
- #define `WM8960_I2C_ADDR` 0x1A  
*WM8960 I2C address.*
- #define `WM8960_I2C_BAUDRATE` (100000U)

- WM8960 I<sub>2</sub>C baudrate.  
• #define **WM8960\_ADC\_MAX\_VOLUME\_vVALUE** 0xFFU  
WM8960 maximum volume value.

## Enumerations

- enum **wm8960\_module\_t** {
   
kWM8960\_ModuleADC = 0,  
kWM8960\_ModuleDAC = 1,  
kWM8960\_ModuleVREF = 2,  
kWM8960\_ModuleHP = 3,  
kWM8960\_ModuleMICB = 4,  
kWM8960\_ModuleMIC = 5,  
kWM8960\_ModuleLineIn = 6,  
kWM8960\_ModuleLineOut = 7,  
kWM8960\_ModuleSpeaker = 8,  
kWM8960\_ModuleOMIX = 9 }

Modules in WM8960 board.

- enum {
   
kWM8960\_HeadphoneLeft = 1,  
kWM8960\_HeadphoneRight = 2,  
kWM8960\_SpeakerLeft = 4,  
kWM8960\_SpeakerRight = 8 }

wm8960 play channel

- enum **wm8960\_play\_source\_t** {
   
kWM8960\_PlaySourcePGA = 1,  
kWM8960\_PlaySourceInput = 2,  
kWM8960\_PlaySourceDAC = 4 }

wm8960 play source

- enum **wm8960\_route\_t** {
   
kWM8960\_RouteBypass = 0,  
kWM8960\_RoutePlayback = 1,  
kWM8960\_RoutePlaybackandRecord = 2,  
kWM8960\_RouteRecord = 5 }

WM8960 data route.

- enum **wm8960\_protocol\_t** {
   
kWM8960\_BusI2S = 2,  
kWM8960\_BusLeftJustified = 1,  
kWM8960\_BusRightJustified = 0,  
kWM8960\_BusPCMA = 3,  
kWM8960\_BusPCMB = 3 | (1 << 4) }

The audio data transfer protocol choice.

- enum **wm8960\_input\_t** {

```

kWM8960_InputClosed = 0,
kWM8960_InputSingleEndedMic = 1,
kWM8960_InputDifferentialMicInput2 = 2,
kWM8960_InputDifferentialMicInput3 = 3,
kWM8960_InputLineINPUT2 = 4,
kWM8960_InputLineINPUT3 = 5 }

 wm8960 input source
• enum {
 kWM8960_AudioSampleRate8KHz = 8000U,
 kWM8960_AudioSampleRate11025Hz = 11025U,
 kWM8960_AudioSampleRate12KHz = 12000U,
 kWM8960_AudioSampleRate16KHz = 16000U,
 kWM8960_AudioSampleRate22050Hz = 22050U,
 kWM8960_AudioSampleRate24KHz = 24000U,
 kWM8960_AudioSampleRate32KHz = 32000U,
 kWM8960_AudioSampleRate44100Hz = 44100U,
 kWM8960_AudioSampleRate48KHz = 48000U,
 kWM8960_AudioSampleRate96KHz = 96000U,
 kWM8960_AudioSampleRate192KHz = 192000U,
 kWM8960_AudioSampleRate384KHz = 384000U }

 audio sample rate definition
• enum {
 kWM8960_AudioBitWidth16bit = 16U,
 kWM8960_AudioBitWidth20bit = 20U,
 kWM8960_AudioBitWidth24bit = 24U,
 kWM8960_AudioBitWidth32bit = 32U }

 audio bit width
• enum wm8960_sysclk_source_t {
 kWM8960_SysClkSourceMclk = 0U,
 kWM8960_SysClkSourceInternalPLL = 1U }

 wm8960 sysclk source

```

## Functions

- **status\_t WM8960\_Init** (**wm8960\_handle\_t** \*handle, const **wm8960\_config\_t** \*config)  
*WM8960 initialize function.*
- **status\_t WM8960\_Deinit** (**wm8960\_handle\_t** \*handle)  
*Deinit the WM8960 codec.*
- **status\_t WM8960\_SetDataRoute** (**wm8960\_handle\_t** \*handle, **wm8960\_route\_t** route)  
*Set audio data route in WM8960.*
- **status\_t WM8960\_SetLeftInput** (**wm8960\_handle\_t** \*handle, **wm8960\_input\_t** input)  
*Set left audio input source in WM8960.*
- **status\_t WM8960\_SetRightInput** (**wm8960\_handle\_t** \*handle, **wm8960\_input\_t** input)  
*Set right audio input source in WM8960.*
- **status\_t WM8960\_SetProtocol** (**wm8960\_handle\_t** \*handle, **wm8960\_protocol\_t** protocol)  
*Set the audio transfer protocol.*

- void [WM8960\\_SetMasterSlave](#) (wm8960\_handle\_t \*handle, bool master)  
*Set WM8960 as master or slave.*
- status\_t [WM8960\\_SetVolume](#) (wm8960\_handle\_t \*handle, wm8960\_module\_t module, uint32\_t volume)  
*Set the volume of different modules in WM8960.*
- uint32\_t [WM8960\\_GetVolume](#) (wm8960\_handle\_t \*handle, wm8960\_module\_t module)  
*Get the volume of different modules in WM8960.*
- status\_t [WM8960\\_SetMute](#) (wm8960\_handle\_t \*handle, wm8960\_module\_t module, bool isEnabled)  
*Mute modules in WM8960.*
- status\_t [WM8960\\_SetModule](#) (wm8960\_handle\_t \*handle, wm8960\_module\_t module, bool isEnabled)  
*Enable/disable expected devices.*
- status\_t [WM8960\\_SetPlay](#) (wm8960\_handle\_t \*handle, uint32\_t playSource)  
*SET the WM8960 play source.*
- status\_t [WM8960\\_ConfigDataFormat](#) (wm8960\_handle\_t \*handle, uint32\_t sysclk, uint32\_t sample\_rate, uint32\_t bits)  
*Configure the data format of audio data.*
- status\_t [WM8960\\_SetJackDetect](#) (wm8960\_handle\_t \*handle, bool isEnabled)  
*Enable/disable jack detect feature.*
- status\_t [WM8960\\_WriteReg](#) (wm8960\_handle\_t \*handle, uint8\_t reg, uint16\_t val)  
*Write register to WM8960 using I2C.*
- status\_t [WM8960\\_ReadReg](#) (uint8\_t reg, uint16\_t \*val)  
*Read register from WM8960 using I2C.*
- status\_t [WM8960\\_ModifyReg](#) (wm8960\_handle\_t \*handle, uint8\_t reg, uint16\_t mask, uint16\_t val)  
*Modify some bits in the register using I2C.*

## Driver version

- #define [FSL\\_WM8960\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 2, 0))  
*CLOCK driver version 2.2.0.*

### 43.7.2 Data Structure Documentation

#### 43.7.2.1 struct [wm8960\\_audio\\_format\\_t](#)

##### Data Fields

- uint32\_t [mclk\\_HZ](#)  
*master clock frequency*
- uint32\_t [sampleRate](#)  
*sample rate*
- uint32\_t [bitWidth](#)  
*bit width*

### 43.7.2.2 struct wm8960\_master\_sysclk\_config\_t

#### Data Fields

- `wm8960_sysclk_source_t sysclkSource`  
*sysclk source*
- `uint32_t sysclkFreq`  
*PLL output frequency value.*

### 43.7.2.3 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`

### 43.7.2.4 struct wm8960\_handle\_t

#### Data Fields

- `const wm8960_config_t * config`  
*wm8904 config pointer*
- `uint8_t i2cHandle [WM8960_I2C_HANDLER_SIZE]`  
*i2c handle*

### 43.7.3 Macro Definition Documentation

#### 43.7.3.1 #define WM8960\_LINVOL 0x0U

#### 43.7.3.2 #define WM8960\_I2C\_ADDR 0x1A

### 43.7.4 Enumeration Type Documentation

#### 43.7.4.1 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.

#### 43.7.4.2 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

#### 43.7.4.3 enum wm8960\_play\_source\_t

Enumerator

*kWM8960\_PlaySourcePGA* wm8960 play source PGA  
*kWM8960\_PlaySourceInput* wm8960 play source Input  
*kWM8960\_PlaySourceDAC* wm8960 play source DAC

#### 43.7.4.4 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.

#### 43.7.4.5 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.

#### 43.7.4.6 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.

#### 43.7.4.7 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.

#### 43.7.4.8 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

#### 43.7.4.9 enum **wm8960\_sysclk\_source\_t**

Enumerator

*kWM8960\_SysClkSourceMclk* sysclk source from external MCLK  
*kWM8960\_SysClkSourceInternalPLL* sysclk source from internal PLL

### 43.7.5 Function Documentation

#### 43.7.5.1 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. |

#### 43.7.5.2 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. |
|---------------|----------------------------------|

#### 43.7.5.3 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. |

#### 43.7.5.4 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.      |

#### 43.7.5.5 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.      |

#### 43.7.5.6 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. |

#### 43.7.5.7 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. |

#### 43.7.5.8 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.                                   |

#### 43.7.5.9 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.

#### 43.7.5.10 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. |

#### 43.7.5.11 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. |

#### 43.7.5.12 status\_t WM8960\_SetPlay ( *wm8960\_handle\_t \* handle, uint32\_t playSource* )

Parameters

|                   |                                                                                                                                                                                                      |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>     | WM8960 handle structure.                                                                                                                                                                             |
| <i>playSource</i> | play source , can be a value combine of kWM8960_ModuleHeadphoneSourcePG-A, kWM8960_ModuleHeadphoneSourceDAC, kWM8960_ModulePlaySourceInput, kWM8960_ModulePlayMonoRight, kWM8960_ModulePlayMonoLeft. |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise..

**43.7.5.13 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).                                                      |

**43.7.5.14 status\_t WM8960\_SetJackDetect ( *wm8960\_handle\_t \* handle, bool isEnabled* )**

Parameters

|                  |                            |
|------------------|----------------------------|
| <i>handle</i>    | WM8960 handle structure.   |
| <i>isEnabled</i> | Enable or disable moudles. |

**43.7.5.15 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. |

**43.7.5.16 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.               |

**43.7.5.17 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.                                          |

## 43.7.6 WM8960 Adapter

### 43.7.6.1 Overview

The wm8960 adapter provides a codec unify control interface.

#### Macros

- #define `HAL_CODEC_WM8960_HANDLER_SIZE` (`WM8960_I2C_HANDLER_SIZE + 4`)  
*codec handler size*

#### Functions

- `status_t HAL_CODEC_WM8960_Init` (void \*handle, void \*config)  
*Codec initialization.*
- `status_t HAL_CODEC_WM8960_Deinit` (void \*handle)  
*Codec de-initilization.*
- `status_t HAL_CODEC_WM8960_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- `status_t HAL_CODEC_WM8960_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- `status_t HAL_CODEC_WM8960_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- `status_t HAL_CODEC_WM8960_SetPower` (void \*handle, uint32\_t module, bool powerOn)  
*set audio codec module power.*
- `status_t HAL_CODEC_WM8960_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- `status_t HAL_CODEC_WM8960_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- `status_t HAL_CODEC_WM8960_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- `status_t HAL_CODEC_WM8960_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*
- static `status_t HAL_CODEC_Init` (void \*handle, void \*config)  
*Codec initilization.*
- static `status_t HAL_CODEC_Deinit` (void \*handle)  
*Codec de-initilization.*
- static `status_t HAL_CODEC_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- static `status_t HAL_CODEC_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- static `status_t HAL_CODEC_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- static `status_t HAL_CODEC_SetPower` (void \*handle, uint32\_t module, bool powerOn)

- static `status_t HAL_CODEC_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- static `status_t HAL_CODEC_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- static `status_t HAL_CODEC_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- static `status_t HAL_CODEC_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

### 43.7.6.2 Function Documentation

#### 43.7.6.2.1 `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.

#### 43.7.6.2.2 `status_t HAL_CODEC_WM8960_Deinit( void * handle )`

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

#### 43.7.6.2.3 `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.

#### 43.7.6.2.4 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.

#### 43.7.6.2.5 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.

#### 43.7.6.2.6 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.

#### 43.7.6.2.7 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.

#### 43.7.6.2.8 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.

#### 43.7.6.2.9 status\_t HAL\_CODEC\_WM8960\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 43.7.6.2.10 status\_t HAL\_CODEC\_WM8960\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* )

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

#### 43.7.6.2.11 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.

#### 43.7.6.2.12 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.

#### 43.7.6.2.13 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.

#### 43.7.6.2.14 static status\_t HAL\_CODEC\_SetVolume( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

Returns

kStatus\_Success is success, else configure failed.

#### 43.7.6.2.15 static status\_t HAL\_CODEC\_SetMute( void \* *handle*, uint32\_t *playChannel*, bool *isMute* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

Returns

kStatus\_Success is success, else configure failed.

#### 43.7.6.2.16 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.

#### 43.7.6.2.17 static status\_t HAL\_CODEC\_SetRecord ( void \* *handle*, uint32\_t *recordSource* ) [inline], [static]

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 43.7.6.2.18 static status\_t HAL\_CODEC\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* ) [inline], [static]

Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

Returns

kStatus\_Success is success, else configure failed.

**43.7.6.2.19 static status\_t HAL\_CODEC\_SetPlay ( void \* *handle*, uint32\_t *playSource* ) [inline], [static]**

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

**43.7.6.2.20 static status\_t HAL\_CODEC\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* ) [inline], [static]**

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

## 43.8 WM8904 Driver

### 43.8.1 Overview

The wm8904 driver provides a codec control interface.

## Data Structures

- struct [wm8904\\_fll\\_config\\_t](#)  
*wm8904 fll configuration* [More...](#)
- struct [wm8904\\_audio\\_format\\_t](#)  
*Audio format configuration.* [More...](#)
- struct [wm8904\\_config\\_t](#)  
*Configuration structure of WM8904.* [More...](#)
- struct [wm8904\\_handle\\_t](#)  
*wm8904 codec handler* [More...](#)

## Macros

- #define [WM8904\\_I2C\\_HANDLER\\_SIZE](#) (CODEC\_I2C\_MASTER\_HANDLER\_SIZE)  
*wm8904 handle size*
- #define [WM8904\\_DEBUG\\_REGISTER](#) 0  
*wm8904 debug macro*
- #define [WM8904\\_RESET](#) (0x00)  
*WM8904 register map.*
- #define [WM8904\\_I2C\\_ADDRESS](#) (0x1A)  
*WM8904 I2C address.*
- #define [WM8904\\_I2C\\_BITRATE](#) (400000U)  
*WM8904 I2C bit rate.*
- #define [WM8904\\_MAP\\_HEADPHONE\\_LINEOUT\\_MAX\\_VOLUME](#) 0x3FU  
*WM8904 maximum volume.*

## Enumerations

- enum {
   
*kStatus\_WM8904\_Success* = 0x0,  
*kStatus\_WM8904\_Fail* = 0x1
 }
   
*WM8904 status return codes.*
- enum {
   
*kWM8904\_LRCPolarityNormal* = 0U,  
*kWM8904\_LRCPolarityInverted* = 1U << 4U
 }
   
*WM8904 lrc polarity.*
- enum [wm8904\\_module\\_t](#) {

- ```
kWM8904_ModuleADC = 0,
kWM8904_ModuleDAC = 1,
kWM8904_ModulePGA = 2,
kWM8904_ModuleHeadphone = 3,
kWM8904_ModuleLineout = 4 }
```

wm8904 module value
- enum

```
wm8904 play channel
```
- enum `wm8904_timeslot_t` {

```
kWM8904_TimeSlot0 = 0U,
kWM8904_TimeSlot1 = 1U }
```

WM8904 time slot.
- enum `wm8904_protocol_t` {

```
kWM8904_ProtocolI2S = 0x2,
kWM8904_ProtocolLeftJustified = 0x1,
kWM8904_ProtocolRightJustified = 0x0,
kWM8904_ProtocolPCMA = 0x3,
kWM8904_ProtocolPCMB = 0x3 | (1 << 4) }
```

The audio data transfer protocol.
- enum `wm8904_fs_ratio_t` {

```
kWM8904_FsRatio64X = 0x0,
kWM8904_FsRatio128X = 0x1,
kWM8904_FsRatio192X = 0x2,
kWM8904_FsRatio256X = 0x3,
kWM8904_FsRatio384X = 0x4,
kWM8904_FsRatio512X = 0x5,
kWM8904_FsRatio768X = 0x6,
kWM8904_FsRatio1024X = 0x7,
kWM8904_FsRatio1408X = 0x8,
kWM8904_FsRatio1536X = 0x9 }
```

The SYSCLK / fs ratio.
- enum `wm8904_sample_rate_t` {

```
kWM8904_SampleRate8kHz = 0x0,
kWM8904_SampleRate12kHz = 0x1,
kWM8904_SampleRate16kHz = 0x2,
kWM8904_SampleRate24kHz = 0x3,
kWM8904_SampleRate32kHz = 0x4,
kWM8904_SampleRate48kHz = 0x5,
kWM8904_SampleRate11025Hz = 0x6,
kWM8904_SampleRate22050Hz = 0x7,
kWM8904_SampleRate44100Hz = 0x8 }
```

Sample rate.
- enum `wm8904_bit_width_t` {

```
kWM8904_BitWidth16 = 0x0,
kWM8904_BitWidth20 = 0x1,
kWM8904_BitWidth24 = 0x2,
```

```

kWM8904_BitWidth32 = 0x3 }

Bit width.
• enum {
    kWM8904_RecordSourceDifferentialLine = 1U,
    kWM8904_RecordSourceLineInput = 2U,
    kWM8904_RecordSourceDifferentialMic = 4U,
    kWM8904_RecordSourceDigitalMic = 8U }
    wm8904 record source
• enum {
    kWM8904_RecordChannelLeft1 = 1U,
    kWM8904_RecordChannelLeft2 = 2U,
    kWM8904_RecordChannelLeft3 = 4U,
    kWM8904_RecordChannelRight1 = 1U,
    kWM8904_RecordChannelRight2 = 2U,
    kWM8904_RecordChannelRight3 = 4U,
    kWM8904_RecordChannelDifferentialPositive1 = 1U,
    kWM8904_RecordChannelDifferentialPositive2 = 2U,
    kWM8904_RecordChannelDifferentialPositive3 = 4U,
    kWM8904_RecordChannelDifferentialNegative1 = 8U,
    kWM8904_RecordChannelDifferentialNegative2 = 16U,
    kWM8904_RecordChannelDifferentialNegative3 = 32U }
    wm8904 record channel
• enum {
    kWM8904_PlaySourcePGA = 1U,
    kWM8904_PlaySourceDAC = 4U }
    wm8904 play source
• enum wm8904\_sys\_clk\_source\_t {
    kWM8904_SysClkSourceMCLK = 0U,
    kWM8904_SysClkSourceFLL = 1U << 14 }
    wm8904 system clock source
• enum wm8904\_fll\_clk\_source\_t { kWM8904_FLLClkSourceMCLK = 0U }
    wm8904 fll clock source

```

Functions

- [status_t WM8904_WriteRegister \(wm8904_handle_t *handle, uint8_t reg, uint16_t value\)](#)
WM8904 write register.
- [status_t WM8904_ReadRegister \(wm8904_handle_t *handle, uint8_t reg, uint16_t *value\)](#)
WM8904 write register.
- [status_t WM8904_ModifyRegister \(wm8904_handle_t *handle, uint8_t reg, uint16_t mask, uint16_t value\)](#)
WM8904 modify register.
- [status_t WM8904_Init \(wm8904_handle_t *handle, \[wm8904_config_t\]\(#\) *wm8904Config\)](#)
Initializes WM8904.
- [status_t WM8904_Deinit \(wm8904_handle_t *handle\)](#)
Deinitializes the WM8904 codec.
- [void WM8904_GetDefaultConfig \(\[wm8904_config_t\]\(#\) *config\)](#)

Fills the configuration structure with default values.

- **status_t WM8904_SetMasterSlave** (`wm8904_handle_t *handle, bool master`)

Sets WM8904 as master or slave.
- **status_t WM8904_SetMasterClock** (`wm8904_handle_t *handle, uint32_t sysclk, uint32_t sampleRate, uint32_t bitWidth`)

Sets WM8904 master clock configuration.
- **status_t WM8904_SetFLLConfig** (`wm8904_handle_t *handle, wm8904_fll_config_t *config`)

WM8904 set PLL configuration This function will enable the GPIO1 FLL clock output function, so user can see the generated fll output clock frequency from WM8904 GPIO1.
- **status_t WM8904_SetProtocol** (`wm8904_handle_t *handle, wm8904_protocol_t protocol`)

Sets the audio data transfer protocol.
- **status_t WM8904_SetAudioFormat** (`wm8904_handle_t *handle, uint32_t sysclk, uint32_t sampleRate, uint32_t bitWidth`)

Sets the audio data format.
- **status_t WM8904_CheckAudioFormat** (`wm8904_handle_t *handle, wm8904_audio_format_t *format, uint32_t mclkFreq`)

check and update the audio data format.
- **status_t WM8904_SetVolume** (`wm8904_handle_t *handle, uint16_t volumeLeft, uint16_t volumeRight`)

Sets the module output volume.
- **status_t WM8904_SetMute** (`wm8904_handle_t *handle, bool muteLeft, bool muteRight`)

Sets the headphone output mute.
- **status_t WM8904_SelectLRCPolarity** (`wm8904_handle_t *handle, uint32_t polarity`)

Select LRC polarity.
- **status_t WM8904_EnableDACTDMMMode** (`wm8904_handle_t *handle, wm8904_timeslot_t timeSlot`)

Enable WM8904 DAC time slot.
- **status_t WM8904_EnableADCTDMMMode** (`wm8904_handle_t *handle, wm8904_timeslot_t timeSlot`)

Enable WM8904 ADC time slot.
- **status_t WM8904_SetModulePower** (`wm8904_handle_t *handle, wm8904_module_t module, bool isEnabled`)

SET the module output power.
- **status_t WM8904_SetDACVolume** (`wm8904_handle_t *handle, uint8_t volume`)

SET the DAC module volume.
- **status_t WM8904_SetChannelVolume** (`wm8904_handle_t *handle, uint32_t channel, uint32_t volume`)

Sets the channel output volume.
- **status_t WM8904_SetRecord** (`wm8904_handle_t *handle, uint32_t recordSource`)

SET the WM8904 record source.
- **status_t WM8904_SetRecordChannel** (`wm8904_handle_t *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel`)

SET the WM8904 record source.
- **status_t WM8904_SetPlay** (`wm8904_handle_t *handle, uint32_t playSource`)

SET the WM8904 play source.
- **status_t WM8904_SetChannelMute** (`wm8904_handle_t *handle, uint32_t channel, bool isMute`)

Sets the channel mute.

Driver version

- #define `FSL_WM8904_DRIVER_VERSION` (`MAKE_VERSION(2, 5, 0)`)
WM8904 driver version 2.5.0.

43.8.2 Data Structure Documentation

43.8.2.1 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

43.8.2.2 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.

43.8.2.3 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

43.8.2.4 struct `wm8904_handle_t`

Data Fields

- `wm8904_config_t * config`
wm8904 config pointer
- `uint8_t i2cHandle [WM8904_I2C_HANDLER_SIZE]`
i2c handle

43.8.3 Macro Definition Documentation

43.8.3.1 `#define FSL_WM8904_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))`

43.8.3.2 `#define WM8904_I2C_ADDRESS (0x1A)`

43.8.3.3 `#define WM8904_I2C_BITRATE (400000U)`

43.8.4 Enumeration Type Documentation

43.8.4.1 anonymous enum

Enumerator

kStatus_WM8904_Success Success.
kStatus_WM8904_Fail Failure.

43.8.4.2 anonymous enum

Enumerator

kWM8904_LRCPolarityNormal LRC polarity normal.
kWM8904_LRCPolarityInverted LRC polarity inverted.

43.8.4.3 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

43.8.4.4 anonymous enum

43.8.4.5 enum wm8904_timeslot_t

Enumerator

- kWM8904_TimeSlot0* time slot0
- kWM8904_TimeSlot1* time slot1

43.8.4.6 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.

43.8.4.7 enum wm8904_fs_ratio_t

Enumerator

- kWM8904_FsRatio64X* SYSCLK is $64 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio128X* SYSCLK is $128 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio192X* SYSCLK is $192 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio256X* SYSCLK is $256 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio384X* SYSCLK is $384 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio512X* SYSCLK is $512 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio768X* SYSCLK is $768 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio1024X* SYSCLK is $1024 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio1408X* SYSCLK is $1408 * \text{sample rate} * \text{frame width}$.
- kWM8904_FsRatio1536X* SYSCLK is $1536 * \text{sample rate} * \text{frame width}$.

43.8.4.8 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

43.8.4.9 enum `wm8904_bit_width_t`

Enumerator

- kWM8904_BitWidth16* 16 bits
- kWM8904_BitWidth20* 20 bits
- kWM8904_BitWidth24* 24 bits
- kWM8904_BitWidth32* 32 bits

43.8.4.10 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

43.8.4.11 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

43.8.4.12 anonymous enum

Enumerator

- kWM8904_PlaySourcePGA* play source PGA, bypass ADC
- kWM8904_PlaySourceDAC* play source Input3

43.8.4.13 enum `wm8904_sys_clk_source_t`

Enumerator

- kWM8904_SysClkSourceMCLK* wm8904 system clock soure from MCLK
- kWM8904_SysClkSourceFLL* wm8904 system clock soure from FLL

43.8.4.14 enum `wm8904_fll_clk_source_t`

Enumerator

- kWM8904_FLLClkSourceMCLK* wm8904 FLL clock source from MCLK

43.8.5 Function Documentation

43.8.5.1 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.

43.8.5.2 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.

43.8.5.3 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.

43.8.5.4 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.

43.8.5.5 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.

43.8.5.6 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.
---------------	-----------------------------------

43.8.5.7 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.

43.8.5.8 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.

43.8.5.9 status_t WM8904_SetFLLConfig (*wm8904_handle_t * handle*, *wm8904_fll_config_t * config*)

Parameters

<i>handle</i>	wm8904 handler pointer.
<i>config</i>	FLL configuration pointer.

43.8.5.10 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.

43.8.5.11 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.

43.8.5.12 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.

43.8.5.13 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.

43.8.5.14 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.

43.8.5.15 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.

43.8.5.16 status_t WM8904_EnableDACTDMMode (*wm8904_handle_t * handle, wm8904_timeslot_t timeSlot*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>timeSlot</i>	timeslot number.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

43.8.5.17 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.

43.8.5.18 status_t WM8904_SetModulePower (*wm8904_handle_t * handle,* *wm8904_module_t module, bool isEnabled*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>module</i>	wm8904 module.
<i>isEnabled</i> , <i>true</i>	is power on, false is power down.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

43.8.5.19 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..

43.8.5.20 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.

43.8.5.21 status_t WM8904_SetRecord (*wm8904_handle_t * handle, uint32_t recordSource*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>recordSource</i>	record source , can be a value of kCODEC_ModuleRecordSourceDifferential-Line, kCODEC_ModuleRecordSourceDifferentialMic, kCODEC_ModuleRecord-SourceSingleEndMic, kCODEC_ModuleRecordSourceDigitalMic.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

43.8.5.22 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..

43.8.5.23 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..

43.8.5.24 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..

43.8.6 WM8904 Adapter

43.8.6.1 Overview

The wm8904 adapter provides a codec unify control interface.

Macros

- #define `HAL_CODEC_WM8904_HANDLER_SIZE` (`WM8904_I2C_HANDLER_SIZE + 4`)
codec handler size

Functions

- `status_t HAL_CODEC_WM8904_Init` (void *handle, void *config)
Codec initialization.
- `status_t HAL_CODEC_WM8904_Deinit` (void *handle)
Codec de-initilization.
- `status_t HAL_CODEC_WM8904_SetFormat` (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
set audio data format.
- `status_t HAL_CODEC_WM8904_SetVolume` (void *handle, uint32_t playChannel, uint32_t volume)
set audio codec module volume.
- `status_t HAL_CODEC_WM8904_SetMute` (void *handle, uint32_t playChannel, bool isMute)
set audio codec module mute.
- `status_t HAL_CODEC_WM8904_SetPower` (void *handle, uint32_t module, bool powerOn)
set audio codec module power.
- `status_t HAL_CODEC_WM8904_SetRecord` (void *handle, uint32_t recordSource)
codec set record source.
- `status_t HAL_CODEC_WM8904_SetRecordChannel` (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
codec set record channel.
- `status_t HAL_CODEC_WM8904_SetPlay` (void *handle, uint32_t playSource)
codec set play source.
- `status_t HAL_CODEC_WM8904_ModuleControl` (void *handle, uint32_t cmd, uint32_t data)
codec module control.
- static `status_t HAL_CODEC_Init` (void *handle, void *config)
Codec initilization.
- static `status_t HAL_CODEC_Deinit` (void *handle)
Codec de-initilization.
- static `status_t HAL_CODEC_SetFormat` (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
set audio data format.
- static `status_t HAL_CODEC_SetVolume` (void *handle, uint32_t playChannel, uint32_t volume)
set audio codec module volume.
- static `status_t HAL_CODEC_SetMute` (void *handle, uint32_t playChannel, bool isMute)
set audio codec module mute.
- static `status_t HAL_CODEC_SetPower` (void *handle, uint32_t module, bool powerOn)

- static `status_t HAL_CODEC_SetRecord` (void *handle, uint32_t recordSource)
codec set record source.
- static `status_t HAL_CODEC_SetRecordChannel` (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
codec set record channel.
- static `status_t HAL_CODEC_SetPlay` (void *handle, uint32_t playSource)
codec set play source.
- static `status_t HAL_CODEC_ModuleControl` (void *handle, uint32_t cmd, uint32_t data)
codec module control.

43.8.6.2 Function Documentation

43.8.6.2.1 `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.

43.8.6.2.2 `status_t HAL_CODEC_WM8904_Deinit(void * handle)`

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus_Success is success, else de-initial failed.

43.8.6.2.3 `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.

43.8.6.2.4 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.

43.8.6.2.5 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.

43.8.6.2.6 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.

43.8.6.2.7 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.

43.8.6.2.8 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.

43.8.6.2.9 status_t HAL_CODEC_WM8904_SetPlay (void * *handle*, uint32_t *playSource*)

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

43.8.6.2.10 status_t HAL_CODEC_WM8904_ModuleControl (void * *handle*, uint32_t *cmd*, uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference _codec_module_ctrl_cmd.
<i>data</i>	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

43.8.6.2.11 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.

43.8.6.2.12 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.

43.8.6.2.13 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.

43.8.6.2.14 static status_t HAL_CODEC_SetVolume(void * *handle*, uint32_t *playChannel*, uint32_t *volume*) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

43.8.6.2.15 static status_t HAL_CODEC_SetMute(void * *handle*, uint32_t *playChannel*, bool *isMute*) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>isMute</i>	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

43.8.6.2.16 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.

43.8.6.2.17 static status_t HAL_CODEC_SetRecord (void * *handle*, uint32_t *recordSource*) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

43.8.6.2.18 static status_t HAL_CODEC_SetRecordChannel (void * *handle*, uint32_t *leftRecordChannel*, uint32_t *rightRecordChannel*) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

43.8.6.2.19 static status_t HAL_CODEC_SetPlay (void * *handle*, uint32_t *playSource*) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

43.8.6.2.20 static status_t HAL_CODEC_ModuleControl (void * *handle*, uint32_t *cmd*, uint32_t *data*) [inline], [static]

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference _codec_module_ctrl_cmd.
<i>data</i>	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

Chapter 44

Serial Manager

44.1 Overview

This chapter describes the programming interface of the serial manager component.

The serial manager component provides a series of APIs to operate different serial port types. The port types it supports are UART, USB CDC and SWO.

Modules

- [Serial Port SWO](#)
- [Serial Port USB](#)
- [Serial Port Uart](#)

Data Structures

- struct [serial_manager_config_t](#)
serial manager config structure [More...](#)
- struct [serial_manager_callback_message_t](#)
Callback message structure. [More...](#)

Macros

- #define [SERIAL_MANAGER_NON_BLOCKING_MODE](#) (0U)
Enable or disable serial manager non-blocking mode (1 - enable, 0 - disable)
- #define [SERIAL_MANAGER_RING_BUFFER_FLOWCONTROL](#) (0U)
Enable or ring buffer flow control (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_UART](#) (0U)
Enable or disable uart port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_UART_DMA](#) (0U)
Enable or disable uart dma port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_USBCDC](#) (0U)
Enable or disable USB CDC port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_SWO](#) (0U)
Enable or disable SWO port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_VIRTUAL](#) (0U)
Enable or disable USB CDC virtual port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_RPMSG](#) (0U)
Enable or disable rpmsg port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_SPI_MASTER](#) (0U)
Enable or disable SPI Master port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_SPI_SLAVE](#) (0U)
Enable or disable SPI Slave port (1 - enable, 0 - disable)
- #define [SERIAL_MANAGER_TASK_HANDLE_TX](#) (0U)
Enable or disable SerialManager_Task() handle TX to prevent recursive calling.

- #define **SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE** (1U)
Set the default delay time in ms used by SerialManager_WriteTimeDelay().
- #define **SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE** (1U)
Set the default delay time in ms used by SerialManager_ReadTimeDelay().
- #define **SERIAL_MANAGER_TASK_HANDLE_RX_AVAILABLE_NOTIFY** (0U)
Enable or disable SerialManager_Task() handle RX data available notify.
- #define **SERIAL_MANAGER_WRITE_HANDLE_SIZE** (4U)
Set serial manager write handle size.
- #define **SERIAL_MANAGER_USE_COMMON_TASK** (0U)
SERIAL_PORT_UART_HANDLE_SIZE/SERIAL_PORT_USB_CDC_HANDLE_SIZE + serial manager dedicated size.
- #define **SERIAL_MANAGER_HANDLE_SIZE** (SERIAL_MANAGER_HANDLE_SIZE_TEMP + 12U)
Macro to determine whether use common task.
- #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)
callback function

Enumerations

- enum `serial_port_type_t` {

 `kSerialPort_Uart` = 1U,

 `kSerialPort_UsbCdc`,

 `kSerialPort_Swo`,

 `kSerialPort_Virtual`,

 `kSerialPort_Rpmsg`,

 `kSerialPort_UartDma`,

 `kSerialPort_SpiMaster`,

 `kSerialPort_SpiSlave`,

 `kSerialPort_None` }

 serial port type
- enum `serial_manager_type_t` {

 `kSerialManager_NonBlocking` = 0x0U,

 `kSerialManager_Blocking` = 0x8F41U }

 serial manager type
- enum `serial_manager_status_t` {

 `kStatus_SerialManager_Success` = `kStatus_Success`,

 `kStatus_SerialManager_Error` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 1)`,

 `kStatus_SerialManager_Busy` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 2)`,

 `kStatus_SerialManager_Notify` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 3)`,

 `kStatus_SerialManager_Canceled`,

 `kStatus_SerialManager_HandleConflict` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 5)`,

 `kStatus_SerialManager_RingBufferOverflow`,

 `kStatus_SerialManager_NotConnected` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 7)` }

 serial manager error code

Functions

- `serial_manager_status_t SerialManager_Init (serial_handle_t serialHandle, const serial_manager_config_t *config)`

Initializes a serial manager module with the serial manager handle and the user configuration structure.
- `serial_manager_status_t SerialManager_Deinit (serial_handle_t serialHandle)`

De-initializes the serial manager module instance.
- `serial_manager_status_t SerialManager_OpenWriteHandle (serial_handle_t serialHandle, serial_write_handle_t writeHandle)`

Opens a writing handle for the serial manager module.
- `serial_manager_status_t SerialManager_CloseWriteHandle (serial_write_handle_t writeHandle)`

Closes a writing handle for the serial manager module.
- `serial_manager_status_t SerialManager_OpenReadHandle (serial_handle_t serialHandle, serial_read_handle_t readHandle)`

Opens a reading handle for the serial manager module.
- `serial_manager_status_t SerialManager_CloseReadHandle (serial_read_handle_t readHandle)`

Closes a reading for the serial manager module.

- `serial_manager_status_t SerialManager_WriteBlocking (serial_write_handle_t writeHandle, uint8_t *buffer, uint32_t length)`
Transmits data with the blocking mode.
- `serial_manager_status_t SerialManager_ReadBlocking (serial_read_handle_t readHandle, uint8_t *buffer, uint32_t length)`
Reads data with the blocking mode.
- `serial_manager_status_t SerialManager_EnterLowpower (serial_handle_t serialHandle)`
Prepares to enter low power consumption.
- `serial_manager_status_t SerialManager_ExitLowpower (serial_handle_t serialHandle)`
Restores from low power consumption.
- static bool `SerialManager_needPollingIsr (void)`
Check if need polling ISR.

44.2 Data Structure Documentation

44.2.1 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.

44.2.2 struct serial_manager_callback_message_t

Data Fields

- `uint8_t * buffer`
Transferred buffer.
- `uint32_t length`
Transferred data length.

44.3 Macro Definition Documentation

44.3.1 #define SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE (1U)

44.3.2 #define SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE (1U)

44.3.3 #define SERIAL_MANAGER_USE_COMMON_TASK (0U)

Macro to determine whether use common task.

44.3.4 #define SERIAL_MANAGER_HANDLE_SIZE (SERIAL_MANAGER_HANDLE_SIZE_TEMP + 12U)

Definition of serial manager handle size.

**44.3.5 #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.
-------------	---

**44.3.6 #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.
-------------	---

44.3.7 #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.
-------------	--

44.3.8 #define SERIAL_MANAGER_TASK_PRIORITY (2U)

44.3.9 #define SERIAL_MANAGER_TASK_STACK_SIZE (1000U)

44.4 Enumeration Type Documentation

44.4.1 enum serial_port_type_t

Enumerator

- kSerialPort_Uart* Serial port UART.
- kSerialPort_UsbCdc* Serial port USB CDC.
- kSerialPort_Swo* Serial port SWO.
- kSerialPort_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_None Serial port is none.

44.4.2 enum serial_manager_type_t

Enumerator

kSerialManager_NonBlocking None blocking handle.

kSerialManager_Blocking Blocking handle.

44.4.3 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.

44.5 Function Documentation

44.5.1 serial_manager_status_t SerialManager_Init (serial_handle_t *serialHandle*, const serial_manager_config_t * *config*)

This function configures the Serial Manager module with user-defined settings. The user can configure the configuration structure. The parameter *serialHandle* is a pointer to point to a memory space of size [SERIAL_MANAGER_HANDLE_SIZE](#) allocated by the caller. The Serial Manager module supports three types of serial port, UART (includes UART, USART, LPSCI, LPUART, etc), USB CDC and swo. Please refer to [serial_port_type_t](#) for serial port setting. These three types can be set by using [serial_manager_config_t](#).

Example below shows how to use this API to configure the Serial Manager. For UART,

```
* #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
* static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
* static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
* serial_manager_config_t config;
* serial_port_uart_config_t uartConfig;
* config.type = kSerialPort_Uart;
* config.ringBuffer = &s_ringBuffer[0];
* config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
* uartConfig.instance = 0;
```

```

*   uartConfig.clockRate = 24000000;
*   uartConfig.baudRate = 115200;
*   uartConfig.parityMode = kSerialManager_UartParityDisabled;
*   uartConfig.stopBitCount = kSerialManager_UartOneStopBit;
*   uartConfig.enableRx = 1;
*   uartConfig.enableTx = 1;
*   uartConfig.enableRxRTS = 0;
*   uartConfig.enableTxCTS = 0;
*   config.portConfig = &uartConfig;
*   SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*

```

For USB CDC,

```

*   #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
*   static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
*   static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
*   serial_manager_config_t config;
*   serial_port_usb_cdc_config_t usbCdcConfig;
*   config.type = kSerialPort_UsbCdc;
*   config.ringBuffer = &s_ringBuffer[0];
*   config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
*   usbCdcConfig.controllerIndex =
*       kSerialManager_UsbControllerKhci0;
*   config.portConfig = &usbCdcConfig;
*   SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*

```

Parameters

<i>serialHandle</i>	Pointer to point to a memory space of size SERIAL_MANAGER_HANDLE_SIZE allocated by the caller. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SERIAL_MANAGER_HANDLE_DEFINE(serialHandle) ; or <code>uint32_t serialHandle[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>
<i>config</i>	Pointer to user-defined configuration structure.

Return values

<i>kStatus_SerialManager_Error</i>	An error occurred.
<i>kStatus_SerialManager_Success</i>	The Serial Manager module initialization succeed.

44.5.2 **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.

44.5.3 **serial_manager_status_t SerialManager_OpenWriteHandle (serial_handle_t *serialHandle*, serial_write_handle_t *writeHandle*)**

This function Opens a writing handle for the serial manager module. If the serial manager needs to be used in different tasks, the task should open a dedicated write handle for itself by calling [SerialManager_OpenWriteHandle](#). Since there can only one buffer for transmission for the writing handle at the same time, multiple writing handles need to be opened when the multiple transmission is needed for a task.

Parameters

<i>serialHandle</i>	The serial manager module handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.
<i>writeHandle</i>	The serial manager module writing handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SERIAL_MANAGER_WRITE_HANDLE_DEFINE(writeHandle) ; or <code>uint32_t writeHandle[((SERIAL_MANAGER_WRITE_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>

Return values

<i>kStatus_SerialManager_-Error</i>	An error occurred.
<i>kStatus_SerialManager_-HandleConflict</i>	The writing handle was opened.

<i>kStatus_SerialManager_-Success</i>	The writing handle is opened.
---------------------------------------	-------------------------------

Example below shows how to use this API to write data. For task 1,

```
* static SERIAL_MANAGER_WRITE_HANDLE_DEFINE(s_serialWriteHandle1);
* static uint8_t s_nonBlockingWelcome1[] = "This is non-blocking writing log for task1!\r\n";
* SerialManager_OpenWriteHandle((serial_handle_t)serialHandle
, (serial_write_handle_t)s_serialWriteHandle1);
* SerialManager_InstallTxCallback((serial_write_handle_t)s_serialWriteHandle1,
, Task1_SerialManagerTxCallback,
s_serialWriteHandle1);
* SerialManager_WriteNonBlocking((serial_write_handle_t)s_serialWriteHandle1,
s_nonBlockingWelcome1,
sizeof(s_nonBlockingWelcome1) - 1U);
*
```

For task 2,

```
* static SERIAL_MANAGER_WRITE_HANDLE_DEFINE(s_serialWriteHandle2);
* static uint8_t s_nonBlockingWelcome2[] = "This is non-blocking writing log for task2!\r\n";
* SerialManager_OpenWriteHandle((serial_handle_t)serialHandle
, (serial_write_handle_t)s_serialWriteHandle2);
* SerialManager_InstallTxCallback((serial_write_handle_t)s_serialWriteHandle2,
, Task2_SerialManagerTxCallback,
s_serialWriteHandle2);
* SerialManager_WriteNonBlocking((serial_write_handle_t)s_serialWriteHandle2,
s_nonBlockingWelcome2,
sizeof(s_nonBlockingWelcome2) - 1U);
*
```

44.5.4 serial_manager_status_t SerialManager_CloseWriteHandle (serial_write_handle_t *writeHandle*)

This function Closes a writing handle for the serial manager module.

Parameters

<i>writeHandle</i>	The serial manager module writing handle pointer.
--------------------	---

Return values

<i>kStatus_SerialManager_-Success</i>	The writing handle is closed.
---------------------------------------	-------------------------------

44.5.5 serial_manager_status_t SerialManager_OpenReadHandle (serial_handle_t *serialHandle*, serial_read_handle_t *readHandle*)

This function Opens a reading handle for the serial manager module. The reading handle can not be opened multiple at the same time. The error code kStatus_SerialManager_Busy would be returned when

the previous reading handle is not closed. And there can only be one buffer for receiving for the reading handle at the same time.

Parameters

<i>serialHandle</i>	The serial manager module handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.
<i>readHandle</i>	The serial manager module reading handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SERIAL_MANAGER_READ_HANDLE_DEFINE(readHandle) ; or <code>uint32_t readHandle[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>

Return values

<i>kStatus_SerialManager_Error</i>	An error occurred.
<i>kStatus_SerialManager_Success</i>	The reading handle is opened.
<i>kStatus_SerialManager_Busy</i>	Previous reading handle is not closed.

Example below shows how to use this API to read data.

```
* static SERIAL_MANAGER_READ_HANDLE_DEFINE(s_serialReadHandle);
* SerialManager_OpenReadHandle((serial_handle_t)serialHandle,
*     (serial_read_handle_t)s_serialReadHandle);
* static uint8_t s_nonBlockingBuffer[64];
* SerialManager_InstallRxCallback((serial_read_handle_t)s_serialReadHandle,
*     APP_SerialManagerRxCallback,
*     s_serialReadHandle);
* SerialManager_ReadNonBlocking((serial_read_handle_t)s_serialReadHandle,
*     s_nonBlockingBuffer,
*     sizeof(s_nonBlockingBuffer));
*
```

44.5.6 **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.
---------------------------------------	-------------------------------

44.5.7 `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.

44.5.8 `serial_manager_status_t SerialManager_ReadBlocking (serial_read_handle_t readHandle, uint8_t * buffer, uint32_t length)`

This is a blocking function, which polls the receiving buffer, waits for the receiving buffer to be full. This function receives data using an interrupt method. The interrupt of the hardware could not be disabled. And There can only one buffer for receiving for the reading handle at the same time.

Note

The function `SerialManager_ReadBlocking` and the function `SerialManager_ReadNonBlocking` cannot be used at the same time. And, the function `SerialManager_CancelReading` cannot be used to abort the transmission of this function.

Parameters

<code>readHandle</code>	The serial manager module handle pointer.
<code>buffer</code>	Start address of the data to store the received data.
<code>length</code>	The length of the data to be received.

Return values

<code>kStatus_SerialManager_-Success</code>	Successfully received all data.
<code>kStatus_SerialManager_-Busy</code>	Previous transmission still not finished; data not all received yet.
<code>kStatus_SerialManager_-Error</code>	An error occurred.

44.5.9 `serial_manager_status_t SerialManager_EnterLowpower (serial_handle_t serialHandle)`

This function is used to prepare to enter low power consumption.

Parameters

<code>serialHandle</code>	The serial manager module handle pointer.
---------------------------	---

Return values

<code>kStatus_SerialManager_-Success</code>	Successful operation.
---	-----------------------

44.5.10 `serial_manager_status_t SerialManager_ExitLowpower (serial_handle_t serialHandle)`

This function is used to restore from low power consumption.

Parameters

<i>serialHandle</i>	The serial manager module handle pointer.
---------------------	---

Return values

<i>kStatus_SerialManager_-Success</i>	Successful operation.
---------------------------------------	-----------------------

44.5.11 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.
-------------	------------------

44.6 Serial Port Uart

44.6.1 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

44.6.2 Enumeration Type Documentation

44.6.2.1 enum serial_port_uart_parity_mode_t

Enumerator

- kSerialManager_UartParityDisabled** Parity disabled.
kSerialManager_UartParityEven Parity even enabled.
kSerialManager_UartParityOdd Parity odd enabled.

44.6.2.2 enum serial_port_uart_stop_bit_count_t

Enumerator

- kSerialManager_UartOneStopBit** One stop bit.
kSerialManager_UartTwoStopBit Two stop bits.

44.7 Serial Port USB

44.7.1 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.

44.7.2 Data Structure Documentation

44.7.2.1 struct serial_port_usb_cdc_config_t

Data Fields

- `serial_port_usb_cdc_controller_index_t controllerIndex`
controller index

44.7.3 Enumeration Type Documentation

44.7.3.1 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.

44.7.4 USB Device Configuration

44.8 Serial Port SWO

44.8.1 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

44.8.2 Data Structure Documentation

44.8.2.1 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.

44.8.3 Enumeration Type Documentation

44.8.3.1 enum `serial_port_swo_protocol_t`

Enumerator

`kSerialManager_SwoProtocolManchester` SWO Manchester protocol.
`kSerialManager_SwoProtocolNrz` SWO UART/NRZ protocol.

44.8.4 CODEC Adapter

44.8.4.1 Overview

Enumerations

- enum {
 kCODEC_WM8904,
 kCODEC_WM8960,
 kCODEC_WM8524,
 kCODEC_SGTL5000,
 kCODEC_DA7212,
 kCODEC_CS42888,
 kCODEC_CS42448,
 kCODEC_AK4497,
 kCODEC_AK4458,
 kCODEC_TFA9XXX,
 kCODEC_TFA9896 }
 codec type

44.8.4.2 Enumeration Type Documentation

44.8.4.2.1 anonymous enum

Enumerator

kCODEC_WM8904 wm8904
kCODEC_WM8960 wm8960
kCODEC_WM8524 wm8524
kCODEC_SGTL5000 sgtl5000
kCODEC_DA7212 da7212
kCODEC_CS42888 CS42888.
kCODEC_CS42448 CS42448.
kCODEC_AK4497 AK4497.
kCODEC_AK4458 ak4458
kCODEC_TFA9XXX tfa9xxx
kCODEC_TFA9896 tfa9896

How to Reach Us:

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

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

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, Freescale, the Freescale logo, Kinetis, Processor Expert, and Tower are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, Cortex, Keil, Mbed, Mbed Enabled, and Vision are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2021 NXP B.V.

