

Document Number: MCUXSDKAPIRM  
Rev 2.12.0  
Jul 2022

# MCUXpresso SDK API Reference Manual

NXP Semiconductors



# Contents

## Chapter 1 Introduction

## Chapter 2 Trademarks

## Chapter 3 Architectural Overview

## Chapter 4 Clock Driver

<b>4.1 Overview</b>	7
<b>4.2 Data Structure Documentation</b>	15
4.2.1 struct sim_clock_config_t	15
4.2.2 struct oscer_config_t	15
4.2.3 struct osc_config_t	15
4.2.4 struct mcg_pll_config_t	16
4.2.5 struct mcg_config_t	16
<b>4.3 Macro Definition Documentation</b>	17
4.3.1 MCG_CONFIG_CHECK_PARAM	17
4.3.2 FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL	18
4.3.3 FSL_CLOCK_DRIVER_VERSION	18
4.3.4 DMAMUX_CLOCKS	18
4.3.5 RTC_CLOCKS	18
4.3.6 SPI_CLOCKS	18
4.3.7 SLCD_CLOCKS	19
4.3.8 EWM_CLOCKS	19
4.3.9 AFE_CLOCKS	19
4.3.10 LPUART_CLOCKS	19
4.3.11 ADC16_CLOCKS	19
4.3.12 XBAR_CLOCKS	20
4.3.13 SYSPMU_CLOCKS	20
4.3.14 VREF_CLOCKS	20
4.3.15 DMA_CLOCKS	20
4.3.16 PORT_CLOCKS	20
4.3.17 UART_CLOCKS	21
4.3.18 PIT_CLOCKS	21
4.3.19 RNGA_CLOCKS	21
4.3.20 CRC_CLOCKS	21

<b>Section No.</b>	<b>Title</b>	<b>Page No.</b>
4.3.21	I2C_CLOCKS .....	21
4.3.22	LPTMR_CLOCKS .....	22
4.3.23	TMR_CLOCKS .....	22
4.3.24	PDB_CLOCKS .....	22
4.3.25	FTF_CLOCKS .....	22
4.3.26	CMP_CLOCKS .....	22
4.3.27	SYS_CLK .....	23
<b>4.4</b>	<b>Enumeration Type Documentation</b> .....	<b>23</b>
4.4.1	clock_name_t .....	23
4.4.2	clock_ip_name_t .....	23
4.4.3	osc_mode_t .....	23
4.4.4	_osc_cap_load .....	23
4.4.5	_oscer_enable_mode .....	24
4.4.6	mcg_fll_src_t .....	24
4.4.7	mcg_irc_mode_t .....	24
4.4.8	mcg_dmx32_t .....	24
4.4.9	mcg_drs_t .....	24
4.4.10	mcg_pll_ref_src_t .....	25
4.4.11	mcg_clkout_src_t .....	25
4.4.12	mcg_atm_select_t .....	25
4.4.13	mcg_oscsel_t .....	25
4.4.14	mcg_pll_clk_select_t .....	25
4.4.15	mcg_monitor_mode_t .....	26
4.4.16	_mcg_status .....	26
4.4.17	_mcg_status_flags_t .....	26
4.4.18	_mcg_irclk_enable_mode .....	26
4.4.19	_mcg_pll_enable_mode .....	27
4.4.20	mcg_mode_t .....	27
<b>4.5</b>	<b>Function Documentation</b> .....	<b>27</b>
4.5.1	CLOCK_EnableClock .....	27
4.5.2	CLOCK_DisableClock .....	27
4.5.3	CLOCK_SetEr32kClock .....	28
4.5.4	CLOCK_SetLpuartClock .....	28
4.5.5	CLOCK_SetXbarClock .....	28
4.5.6	CLOCK_SetAfeClkSrc .....	28
4.5.7	CLOCK_SetPllFllSelClock .....	28
4.5.8	CLOCK_SetClkOutClock .....	29
4.5.9	CLOCK_SetAdcTriggerClock .....	29
4.5.10	CLOCK_SetOutDiv .....	29
4.5.11	CLOCK_GetAfeFreq .....	29
4.5.12	CLOCK_GetFreq .....	30
4.5.13	CLOCK_GetCoreSysClkFreq .....	31
4.5.14	CLOCK_GetPlatClkFreq .....	31

Section No.	Title	Page No.
4.5.15	CLOCK_GetBusClkFreq .....	31
4.5.16	CLOCK_GetFlashClkFreq .....	31
4.5.17	CLOCK_GetPllFllSelClkFreq .....	31
4.5.18	CLOCK_GetEr32kClkFreq .....	32
4.5.19	CLOCK_GetOsc0ErClkFreq .....	32
4.5.20	CLOCK_SetSimConfig .....	32
4.5.21	CLOCK_SetSimSafeDivs .....	32
4.5.22	CLOCK_GetOutClkFreq .....	32
4.5.23	CLOCK_GetFllFreq .....	32
4.5.24	CLOCK_GetInternalRefClkFreq .....	33
4.5.25	CLOCK_GetFixedFreqClkFreq .....	33
4.5.26	CLOCK_GetPll0Freq .....	33
4.5.27	CLOCK_SetLowPowerEnable .....	33
4.5.28	CLOCK_SetInternalRefClkConfig .....	34
4.5.29	CLOCK_SetExternalRefClkConfig .....	34
4.5.30	CLOCK_SetFllExtRefDiv .....	35
4.5.31	CLOCK_EnablePll0 .....	35
4.5.32	CLOCK_DisablePll0 .....	35
4.5.33	CLOCK_SetOsc0MonitorMode .....	35
4.5.34	CLOCK_SetRtcOscMonitorMode .....	35
4.5.35	CLOCK_SetPll0MonitorMode .....	36
4.5.36	CLOCK_GetStatusFlags .....	36
4.5.37	CLOCK_ClearStatusFlags .....	36
4.5.38	OSC_SetExtRefClkConfig .....	37
4.5.39	OSC_SetCapLoad .....	37
4.5.40	CLOCK_InitOsc0 .....	37
4.5.41	CLOCK_DeinitOsc0 .....	38
4.5.42	CLOCK_SetXtal0Freq .....	38
4.5.43	CLOCK_SetXtal32Freq .....	38
4.5.44	CLOCK_SetSlowIrcFreq .....	38
4.5.45	CLOCK_SetFastIrcFreq .....	38
4.5.46	CLOCK_TrimInternalRefClk .....	39
4.5.47	CLOCK_GetMode .....	39
4.5.48	CLOCK_SetFeiMode .....	39
4.5.49	CLOCK_SetFeeMode .....	40
4.5.50	CLOCK_SetFbiMode .....	41
4.5.51	CLOCK_SetFbeMode .....	42
4.5.52	CLOCK_SetBlpiMode .....	43
4.5.53	CLOCK_SetBlpeMode .....	44
4.5.54	CLOCK_SetPbeMode .....	44
4.5.55	CLOCK_SetPeeMode .....	45
4.5.56	CLOCK_SetPbiMode .....	45
4.5.57	CLOCK_SetPeiMode .....	45
4.5.58	CLOCK_ExternalModeToFbeModeQuick .....	45
4.5.59	CLOCK_InternalModeToFbiModeQuick .....	46

Section No.	Title	Page No.
4.5.60	CLOCK_BootToFeiMode .....	46
4.5.61	CLOCK_BootToFeeMode .....	47
4.5.62	CLOCK_BootToBlpiMode .....	47
4.5.63	CLOCK_BootToBlpeMode .....	48
4.5.64	CLOCK_BootToPeeMode .....	48
4.5.65	CLOCK_BootToPeiMode .....	49
4.5.66	CLOCK_SetMcgConfig .....	49

<b>4.6</b>	<b>Variable Documentation</b> .....	<b>49</b>
4.6.1	g_xtal0Freq .....	49
4.6.2	g_xtal32Freq .....	50

<b>4.7</b>	<b>Multipurpose Clock Generator (MCG)</b> .....	<b>51</b>
4.7.1	Function description .....	51
4.7.2	Typical use case .....	53
4.7.3	Code Configuration Option .....	56

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

<b>5.1</b>	<b>Overview</b> .....	<b>57</b>
------------	-----------------------	-----------

<b>5.2</b>	<b>Typical use case</b> .....	<b>57</b>
5.2.1	Polling Configuration .....	57
5.2.2	Interrupt Configuration .....	57

<b>5.3</b>	<b>Data Structure Documentation</b> .....	<b>60</b>
5.3.1	struct adc16_config_t .....	60
5.3.2	struct adc16_hardware_compare_config_t .....	61
5.3.3	struct adc16_channel_config_t .....	61

<b>5.4</b>	<b>Macro Definition Documentation</b> .....	<b>61</b>
5.4.1	FSL_ADC16_DRIVER_VERSION .....	61

<b>5.5</b>	<b>Enumeration Type Documentation</b> .....	<b>61</b>
5.5.1	_adc16_channel_status_flags .....	62
5.5.2	_adc16_status_flags .....	62
5.5.3	adc16_channel_mux_mode_t .....	62
5.5.4	adc16_clock_divider_t .....	62
5.5.5	adc16_resolution_t .....	62
5.5.6	adc16_clock_source_t .....	63
5.5.7	adc16_long_sample_mode_t .....	63
5.5.8	adc16_reference_voltage_source_t .....	63
5.5.9	adc16_hardware_average_mode_t .....	63
5.5.10	adc16_hardware_compare_mode_t .....	64

<b>5.6</b>	<b>Function Documentation</b> .....	<b>64</b>
------------	-------------------------------------	-----------

Section No.	Title	Page No.
5.6.1	<a href="#">ADC16_Init</a>	64
5.6.2	<a href="#">ADC16_Deinit</a>	64
5.6.3	<a href="#">ADC16_GetDefaultConfig</a>	64
5.6.4	<a href="#">ADC16_DoAutoCalibration</a>	65
5.6.5	<a href="#">ADC16_SetOffsetValue</a>	65
5.6.6	<a href="#">ADC16_EnableDMA</a>	65
5.6.7	<a href="#">ADC16_EnableHardwareTrigger</a>	66
5.6.8	<a href="#">ADC16_SetChannelMuxMode</a>	66
5.6.9	<a href="#">ADC16_SetHardwareCompareConfig</a>	66
5.6.10	<a href="#">ADC16_SetHardwareAverage</a>	67
5.6.11	<a href="#">ADC16_GetStatusFlags</a>	67
5.6.12	<a href="#">ADC16_ClearStatusFlags</a>	67
5.6.13	<a href="#">ADC16_EnableAsynchronousClockOutput</a>	67
5.6.14	<a href="#">ADC16_SetChannelConfig</a>	68
5.6.15	<a href="#">ADC16_GetChannelConversionValue</a>	68
5.6.16	<a href="#">ADC16_GetChannelStatusFlags</a>	69

## Chapter 6 AFE: Analog Front End Driver

<b>6.1</b>	<b>Overview</b>	<b>70</b>
<b>6.2</b>	<b>Function groups</b>	<b>70</b>
6.2.1	<a href="#">Channel configuration structures</a>	70
6.2.2	<a href="#">User configuration structures</a>	70
6.2.3	<a href="#">AFE Initialization</a>	70
6.2.4	<a href="#">AFE Conversion</a>	71
<b>6.3</b>	<b>Typical use case</b>	<b>71</b>
6.3.1	<a href="#">AFE Initialization</a>	71
6.3.2	<a href="#">AFE Conversion</a>	71
<b>6.4</b>	<b>Data Structure Documentation</b>	<b>74</b>
6.4.1	<a href="#">struct afe_channel_config_t</a>	74
6.4.2	<a href="#">struct afe_config_t</a>	74
<b>6.5</b>	<b>Macro Definition Documentation</b>	<b>75</b>
6.5.1	<a href="#">FSL_AFE_DRIVER_VERSION</a>	75
<b>6.6</b>	<b>Enumeration Type Documentation</b>	<b>75</b>
6.6.1	<a href="#">_afe_channel_status_flag</a>	75
6.6.2	<a href="#">anonymous enum</a>	76
6.6.3	<a href="#">anonymous enum</a>	76
6.6.4	<a href="#">anonymous enum</a>	76
6.6.5	<a href="#">afe_decimator_oversample_ratio_t</a>	76
6.6.6	<a href="#">afe_result_format_t</a>	77

<b>Section No.</b>	<b>Title</b>	<b>Page No.</b>
6.6.7	afe_clock_divider_t .....	77
6.6.8	afe_clock_source_t .....	77
6.6.9	afe_pga_gain_t .....	77
6.6.10	afe_bypass_mode_t .....	78
<b>6.7</b>	<b>Function Documentation .....</b>	<b>78</b>
6.7.1	AFE_Init .....	78
6.7.2	AFE_Deinit .....	78
6.7.3	AFE_GetDefaultConfig .....	78
6.7.4	AFE_SoftwareReset .....	79
6.7.5	AFE_Enable .....	79
6.7.6	AFE_SetChannelConfig .....	79
6.7.7	AFE_GetDefaultChannelConfig .....	80
6.7.8	AFE_GetChannelConversionValue .....	80
6.7.9	AFE_DoSoftwareTriggerChannel .....	80
6.7.10	AFE_GetChannelStatusFlags .....	81
6.7.11	AFE_SetChannelPhaseDelayValue .....	81
6.7.12	AFE_SetChannelPhasetDelayOk .....	82
6.7.13	AFE_EnableChannelInterrupts .....	82
6.7.14	AFE_DisableChannelInterrupts .....	82
6.7.15	AFE_GetEnabledChannelInterrupts .....	83
6.7.16	AFE_EnableChannelDMA .....	83
<b>Chapter 7 CMP: Analog Comparator Driver</b>		
<b>7.1</b>	<b>Overview .....</b>	<b>84</b>
<b>7.2</b>	<b>Typical use case .....</b>	<b>84</b>
7.2.1	Polling Configuration .....	84
7.2.2	Interrupt Configuration .....	84
<b>7.3</b>	<b>Data Structure Documentation .....</b>	<b>86</b>
7.3.1	struct cmp_config_t .....	86
7.3.2	struct cmp_filter_config_t .....	86
7.3.3	struct cmp_dac_config_t .....	87
<b>7.4</b>	<b>Macro Definition Documentation .....</b>	<b>87</b>
7.4.1	FSL_CMP_DRIVER_VERSION .....	87
<b>7.5</b>	<b>Enumeration Type Documentation .....</b>	<b>87</b>
7.5.1	_cmp_interrupt_enable .....	87
7.5.2	_cmp_status_flags .....	87
7.5.3	cmp_hysteresis_mode_t .....	88
7.5.4	cmp_reference_voltage_source_t .....	88
<b>7.6</b>	<b>Function Documentation .....</b>	<b>88</b>

Section No.	Title	Page No.
7.6.1	CMP_Init .....	88
7.6.2	CMP_Deinit .....	88
7.6.3	CMP_Enable .....	90
7.6.4	CMP_GetDefaultConfig .....	90
7.6.5	CMP_SetInputChannels .....	90
7.6.6	CMP_EnableDMA .....	91
7.6.7	CMP_EnableWindowMode .....	91
7.6.8	CMP_SetFilterConfig .....	91
7.6.9	CMP_SetDACConfig .....	91
7.6.10	CMP_EnableInterrupts .....	92
7.6.11	CMP_DisableInterrupts .....	92
7.6.12	CMP_GetStatusFlags .....	92
7.6.13	CMP_ClearStatusFlags .....	92

## Chapter 8 Common Driver

<b>8.1</b>	<b>Overview .....</b>	<b>94</b>
<b>8.2</b>	<b>Macro Definition Documentation .....</b>	<b>96</b>
8.2.1	FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ .....	96
8.2.2	MAKE_STATUS .....	96
8.2.3	MAKE_VERSION .....	97
8.2.4	FSL_COMMON_DRIVER_VERSION .....	97
8.2.5	DEBUG_CONSOLE_DEVICE_TYPE_NONE .....	97
8.2.6	DEBUG_CONSOLE_DEVICE_TYPE_UART .....	97
8.2.7	DEBUG_CONSOLE_DEVICE_TYPE_LPUART .....	97
8.2.8	DEBUG_CONSOLE_DEVICE_TYPE_LPSCI .....	97
8.2.9	DEBUG_CONSOLE_DEVICE_TYPE_USBCDC .....	97
8.2.10	DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM .....	97
8.2.11	DEBUG_CONSOLE_DEVICE_TYPE_IUART .....	97
8.2.12	DEBUG_CONSOLE_DEVICE_TYPE_VUSART .....	97
8.2.13	DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART .....	97
8.2.14	DEBUG_CONSOLE_DEVICE_TYPE_SWO .....	97
8.2.15	DEBUG_CONSOLE_DEVICE_TYPE_QSCI .....	97
8.2.16	ARRAY_SIZE .....	97
<b>8.3</b>	<b>Typedef Documentation .....</b>	<b>97</b>
8.3.1	status_t .....	97
<b>8.4</b>	<b>Enumeration Type Documentation .....</b>	<b>98</b>
8.4.1	_status_groups .....	98
8.4.2	anonymous enum .....	100
<b>8.5</b>	<b>Function Documentation .....</b>	<b>101</b>
8.5.1	SDK_Malloc .....	101

Section No.	Title	Page No.
8.5.2	SDK_Free .....	101
8.5.3	SDK_DelayAtLeastUs .....	101

## Chapter 9 CRC: Cyclic Redundancy Check Driver

<b>9.1</b>	<b>Overview .....</b>	<b>102</b>
<b>9.2</b>	<b>CRC Driver Initialization and Configuration .....</b>	<b>102</b>
<b>9.3</b>	<b>CRC Write Data .....</b>	<b>102</b>
<b>9.4</b>	<b>CRC Get Checksum .....</b>	<b>102</b>
<b>9.5</b>	<b>Comments about API usage in RTOS .....</b>	<b>103</b>
<b>9.6</b>	<b>Data Structure Documentation .....</b>	<b>104</b>
9.6.1	struct crc_config_t .....	104
<b>9.7</b>	<b>Macro Definition Documentation .....</b>	<b>105</b>
9.7.1	FSL_CRC_DRIVER_VERSION .....	105
9.7.2	CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT .....	105
<b>9.8</b>	<b>Enumeration Type Documentation .....</b>	<b>105</b>
9.8.1	crc_bits_t .....	105
9.8.2	crc_result_t .....	105
<b>9.9</b>	<b>Function Documentation .....</b>	<b>105</b>
9.9.1	CRC_Init .....	105
9.9.2	CRC_Deinit .....	106
9.9.3	CRC_GetDefaultConfig .....	106
9.9.4	CRC_WriteData .....	106
9.9.5	CRC_Get32bitResult .....	107
9.9.6	CRC_Get16bitResult .....	107

## Chapter 10 DMA: Direct Memory Access Controller Driver

<b>10.1</b>	<b>Overview .....</b>	<b>108</b>
<b>10.2</b>	<b>Typical use case .....</b>	<b>108</b>
10.2.1	DMA Operation .....	108
<b>10.3</b>	<b>Data Structure Documentation .....</b>	<b>111</b>
10.3.1	struct dma_transfer_config_t .....	111
10.3.2	struct dma_channel_link_config_t .....	112
10.3.3	struct dma_handle_t .....	112
<b>10.4</b>	<b>Macro Definition Documentation .....</b>	<b>113</b>

Section No.	Title	Page No.
10.4.1	<a href="#">FSL_DMA_DRIVER_VERSION</a>	113
<b>10.5</b>	<b>Typedef Documentation</b>	<b>113</b>
10.5.1	<a href="#">dma_callback</a>	113
<b>10.6</b>	<b>Enumeration Type Documentation</b>	<b>113</b>
10.6.1	<a href="#">anonymous enum</a>	113
10.6.2	<a href="#">dma_transfer_size_t</a>	113
10.6.3	<a href="#">dma_modulo_t</a>	113
10.6.4	<a href="#">dma_channel_link_type_t</a>	114
10.6.5	<a href="#">dma_transfer_type_t</a>	114
10.6.6	<a href="#">dma_transfer_options_t</a>	114
10.6.7	<a href="#">dma_addr_increment_t</a>	115
10.6.8	<a href="#">anonymous enum</a>	115
<b>10.7</b>	<b>Function Documentation</b>	<b>115</b>
10.7.1	<a href="#">DMA_Init</a>	115
10.7.2	<a href="#">DMA_Deinit</a>	115
10.7.3	<a href="#">DMA_ResetChannel</a>	115
10.7.4	<a href="#">DMA_SetTransferConfig</a>	116
10.7.5	<a href="#">DMA_SetChannelLinkConfig</a>	116
10.7.6	<a href="#">DMA_SetSourceAddress</a>	117
10.7.7	<a href="#">DMA_SetDestinationAddress</a>	117
10.7.8	<a href="#">DMA_SetTransferSize</a>	117
10.7.9	<a href="#">DMA_SetModulo</a>	118
10.7.10	<a href="#">DMA_EnableCycleSteal</a>	118
10.7.11	<a href="#">DMA_EnableAutoAlign</a>	118
10.7.12	<a href="#">DMA_EnableAsyncRequest</a>	119
10.7.13	<a href="#">DMA_EnableInterrupts</a>	120
10.7.14	<a href="#">DMA_DisableInterrupts</a>	120
10.7.15	<a href="#">DMA_EnableChannelRequest</a>	120
10.7.16	<a href="#">DMA_DisableChannelRequest</a>	120
10.7.17	<a href="#">DMA_TriggerChannelStart</a>	121
10.7.18	<a href="#">DMA_EnableAutoStopRequest</a>	121
10.7.19	<a href="#">DMA_GetRemainingBytes</a>	121
10.7.20	<a href="#">DMA_GetChannelStatusFlags</a>	122
10.7.21	<a href="#">DMA_ClearChannelStatusFlags</a>	123
10.7.22	<a href="#">DMA_CreateHandle</a>	123
10.7.23	<a href="#">DMA_SetCallback</a>	123
10.7.24	<a href="#">DMA_PrepTransferConfig</a>	124
10.7.25	<a href="#">DMA_PrepTransfer</a>	124
10.7.26	<a href="#">DMA_SubmitTransfer</a>	125
10.7.27	<a href="#">DMA_StartTransfer</a>	125
10.7.28	<a href="#">DMA_StopTransfer</a>	126
10.7.29	<a href="#">DMA_AbortTransfer</a>	126

Section No.	Title	Page No.
10.7.30	DMA_HandleIRQ .....	126

## Chapter 11 DMAMUX: Direct Memory Access Multiplexer Driver

<b>11.1</b>	<b>Overview</b> .....	<b>127</b>
<b>11.2</b>	<b>Typical use case</b> .....	<b>127</b>
11.2.1	DMAMUX Operation .....	127
<b>11.3</b>	<b>Macro Definition Documentation</b> .....	<b>127</b>
11.3.1	FSL_DMAMUX_DRIVER_VERSION .....	127
<b>11.4</b>	<b>Function Documentation</b> .....	<b>127</b>
11.4.1	DMAMUX_Init .....	128
11.4.2	DMAMUX_Deinit .....	129
11.4.3	DMAMUX_EnableChannel .....	129
11.4.4	DMAMUX_DisableChannel .....	129
11.4.5	DMAMUX_SetSource .....	130
11.4.6	DMAMUX_EnablePeriodTrigger .....	130
11.4.7	DMAMUX_DisablePeriodTrigger .....	130

## Chapter 12 EWM: External Watchdog Monitor Driver

<b>12.1</b>	<b>Overview</b> .....	<b>131</b>
<b>12.2</b>	<b>Typical use case</b> .....	<b>131</b>
<b>12.3</b>	<b>Data Structure Documentation</b> .....	<b>132</b>
12.3.1	struct ewm_config_t .....	132
<b>12.4</b>	<b>Macro Definition Documentation</b> .....	<b>132</b>
12.4.1	FSL_EWM_DRIVER_VERSION .....	132
<b>12.5</b>	<b>Enumeration Type Documentation</b> .....	<b>132</b>
12.5.1	ewm_lpo_clock_source_t .....	132
12.5.2	_ewm_interrupt_enable_t .....	133
12.5.3	_ewm_status_flags_t .....	133
<b>12.6</b>	<b>Function Documentation</b> .....	<b>133</b>
12.6.1	EWM_Init .....	133
12.6.2	EWM_Deinit .....	133
12.6.3	EWM_GetDefaultConfig .....	134
12.6.4	EWM_EnableInterrupts .....	134
12.6.5	EWM_DisableInterrupts .....	134
12.6.6	EWM_GetStatusFlags .....	135
12.6.7	EWM_Refresh .....	135

Section No.	Title	Page No.
<b>Chapter 13 C90TFS Flash Driver</b>		
<b>13.1</b>	<b>Overview</b>	<b>137</b>
<b>13.2</b>	<b>Ftftx FLASH Driver</b>	<b>138</b>
13.2.1	Overview	138
13.2.2	Data Structure Documentation	140
13.2.3	Macro Definition Documentation	141
13.2.4	Enumeration Type Documentation	141
13.2.5	Function Documentation	142
<b>13.3</b>	<b>Ftftx CACHE Driver</b>	<b>157</b>
13.3.1	Overview	157
13.3.2	Data Structure Documentation	157
13.3.3	Enumeration Type Documentation	158
13.3.4	Function Documentation	158
<b>13.4</b>	<b>Ftftx FLEXNVM Driver</b>	<b>161</b>
13.4.1	Overview	161
13.4.2	Data Structure Documentation	163
13.4.3	Enumeration Type Documentation	163
13.4.4	Function Documentation	163
<b>13.5</b>	<b>ftfx feature</b>	<b>177</b>
13.5.1	Overview	177
13.5.2	Macro Definition Documentation	177
13.5.3	ftfx adapter	178
<b>13.6</b>	<b>ftfx controller</b>	<b>179</b>
13.6.1	Overview	179
13.6.2	Data Structure Documentation	182
13.6.3	Macro Definition Documentation	184
13.6.4	Enumeration Type Documentation	184
13.6.5	Function Documentation	186
13.6.6	ftfx utilities	198

## Chapter 14 GPIO: General-Purpose Input/Output Driver

<b>14.1</b>	<b>Overview</b>	<b>199</b>
<b>14.2</b>	<b>Data Structure Documentation</b>	<b>199</b>
14.2.1	struct gpio_pin_config_t	200
<b>14.3</b>	<b>Macro Definition Documentation</b>	<b>200</b>
14.3.1	FSL_GPIO_DRIVER_VERSION	200

Section No.	Title	Page No.
<b>14.4 Enumeration Type Documentation</b>		<b>200</b>
14.4.1 gpio_pin_direction_t		200
14.4.2 gpio_checker_attribute_t		200
<b>14.5 GPIO Driver</b>		<b>202</b>
14.5.1 Overview		202
14.5.2 Typical use case		202
14.5.3 Function Documentation		203
<b>14.6 FGPIO Driver</b>		<b>207</b>
14.6.1 Overview		207
14.6.2 Typical use case		207
14.6.3 Function Documentation		208
<b>Chapter 15 I2C: Inter-Integrated Circuit Driver</b>		
<b>15.1 Overview</b>		<b>212</b>
<b>15.2 I2C Driver</b>		<b>213</b>
15.2.1 Overview		213
15.2.2 Typical use case		213
15.2.3 Data Structure Documentation		218
15.2.4 Macro Definition Documentation		222
15.2.5 Typedef Documentation		222
15.2.6 Enumeration Type Documentation		222
15.2.7 Function Documentation		224
<b>15.3 I2C DMA Driver</b>		<b>238</b>
15.3.1 Overview		238
15.3.2 Data Structure Documentation		238
15.3.3 Macro Definition Documentation		239
15.3.4 Typedef Documentation		239
15.3.5 Function Documentation		239
<b>15.4 I2C FreeRTOS Driver</b>		<b>242</b>
15.4.1 Overview		242
15.4.2 Macro Definition Documentation		242
15.4.3 Function Documentation		242
<b>15.5 I2C CMSIS Driver</b>		<b>245</b>
15.5.1 I2C CMSIS Driver		245
<b>Chapter 16 IRTC: IRTC Driver</b>		
<b>16.1 Overview</b>		<b>247</b>

Section No.	Title	Page No.
<b>16.2 Data Structure Documentation</b>		<b>250</b>
16.2.1 struct irtc_datetime_t		251
16.2.2 struct irtc_daylight_time_t		251
16.2.3 struct irtc_tamper_config_t		252
16.2.4 struct irtc_config_t		252
<b>16.3 Macro Definition Documentation</b>		<b>252</b>
16.3.1 FSL_IRTC_DRIVER_VERSION		252
<b>16.4 Enumeration Type Documentation</b>		<b>252</b>
16.4.1 irtc_filter_clock_source_t		252
16.4.2 irtc_tamper_pins_t		253
16.4.3 irtc_interrupt_enable_t		253
16.4.4 irtc_status_flags_t		253
16.4.5 irtc_alarm_match_t		254
16.4.6 irtc_osc_cap_load_t		254
16.4.7 irtc_clockout_sel_t		255
<b>16.5 Function Documentation</b>		<b>255</b>
16.5.1 IRTC_Init		255
16.5.2 IRTC_Deinit		255
16.5.3 IRTC_GetDefaultConfig		255
16.5.4 IRTC_SetDatetime		256
16.5.5 IRTC_GetDatetime		256
16.5.6 IRTC_SetAlarm		256
16.5.7 IRTC_GetAlarm		257
16.5.8 IRTC_EnableInterrupts		257
16.5.9 IRTC_DisableInterrupts		257
16.5.10 IRTC_GetEnabledInterrupts		257
16.5.11 IRTC_GetStatusFlags		258
16.5.12 IRTC_ClearStatusFlags		258
16.5.13 IRTC_SetOscCapLoad		258
16.5.14 IRTC_SetWriteProtection		259
16.5.15 IRTC_Reset		259
16.5.16 IRTC_Enable32kClkDuringRegisterWrite		259
16.5.17 IRTC_ConfigClockOut		260
16.5.18 IRTC_GetTamperStatusFlag		261
16.5.19 IRTC_ClearTamperStatusFlag		261
16.5.20 IRTC_SetTamperConfigurationOver		261
16.5.21 IRTC_SetDaylightTime		261
16.5.22 IRTC_GetDaylightTime		262
16.5.23 IRTC_SetCoarseCompensation		262
16.5.24 IRTC_SetFineCompensation		262
16.5.25 IRTC_SetTamperParams		263

Section No.	Title	Page No.
<b>Chapter 17 LLWU: Low-Leakage Wakeup Unit Driver</b>		
<b>17.1</b>	<b>Overview</b>	<b>264</b>
<b>17.2</b>	<b>External wakeup pins configurations</b>	<b>264</b>
<b>17.3</b>	<b>Internal wakeup modules configurations</b>	<b>264</b>
<b>17.4</b>	<b>Digital pin filter for external wakeup pin configurations</b>	<b>264</b>
<b>17.5</b>	<b>Data Structure Documentation</b>	<b>265</b>
17.5.1	struct llwu_version_id_t	266
17.5.2	struct llwu_param_t	266
17.5.3	struct llwu_external_pin_filter_mode_t	266
<b>17.6</b>	<b>Macro Definition Documentation</b>	<b>267</b>
17.6.1	FSL_LLWU_DRIVER_VERSION	267
<b>17.7</b>	<b>Enumeration Type Documentation</b>	<b>267</b>
17.7.1	llwu_external_pin_mode_t	267
17.7.2	llwu_pin_filter_mode_t	267
<b>17.8</b>	<b>Function Documentation</b>	<b>267</b>
17.8.1	LLWU_GetVersionId	267
17.8.2	LLWU_GetParam	268
17.8.3	LLWU_SetExternalWakeupsPinMode	269
17.8.4	LLWU_GetExternalWakeupsPinFlag	269
17.8.5	LLWU_ClearExternalWakeupsPinFlag	269
17.8.6	LLWU_EnableInternalModuleInterruptWakeup	270
17.8.7	LLWU_GetInternalWakeupsModuleFlag	270
17.8.8	LLWU_EnableInternalModuleDmaRequestWakeup	270
17.8.9	LLWU_SetPinFilterMode	271
17.8.10	LLWU_GetPinFilterFlag	271
17.8.11	LLWU_ClearPinFilterFlag	271
17.8.12	LLWU_SetResetPinMode	272
<b>Chapter 18 LPTMR: Low-Power Timer</b>		
<b>18.1</b>	<b>Overview</b>	<b>273</b>
<b>18.2</b>	<b>Function groups</b>	<b>273</b>
18.2.1	Initialization and deinitialization	273
18.2.2	Timer period Operations	273
18.2.3	Start and Stop timer operations	273
18.2.4	Status	274
18.2.5	Interrupt	274

Section No.	Title	Page No.
<b>18.3 Typical use case .....</b>		<b>274</b>
18.3.1 LPTMR tick example .....		274
<b>18.4 Data Structure Documentation .....</b>		<b>276</b>
18.4.1 struct lptmr_config_t .....		276
<b>18.5 Enumeration Type Documentation .....</b>		<b>277</b>
18.5.1 lptmr_pin_select_t .....		277
18.5.2 lptmr_pin_polarity_t .....		277
18.5.3 lptmr_timer_mode_t .....		277
18.5.4 lptmr_prescaler_glitch_value_t .....		277
18.5.5 lptmr_prescaler_clock_select_t .....		278
18.5.6 lptmr_interrupt_enable_t .....		278
18.5.7 lptmr_status_flags_t .....		278
<b>18.6 Function Documentation .....</b>		<b>278</b>
18.6.1 LPTMR_Init .....		278
18.6.2 LPTMR_Deinit .....		279
18.6.3 LPTMR_GetDefaultConfig .....		279
18.6.4 LPTMR_EnableInterrupts .....		279
18.6.5 LPTMR_DisableInterrupts .....		279
18.6.6 LPTMR_GetEnabledInterrupts .....		280
18.6.7 LPTMR_GetStatusFlags .....		280
18.6.8 LPTMR_ClearStatusFlags .....		280
18.6.9 LPTMR_SetTimerPeriod .....		281
18.6.10 LPTMR_GetCurrentTimerCount .....		281
18.6.11 LPTMR_StartTimer .....		281
18.6.12 LPTMR_StopTimer .....		282

## Chapter 19 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

<b>19.1 Overview .....</b>	<b>283</b>
<b>19.2 LPUART Driver .....</b>	<b>284</b>
19.2.1 Overview .....	284
19.2.2 Typical use case .....	284
19.2.3 Data Structure Documentation .....	289
19.2.4 Macro Definition Documentation .....	291
19.2.5 Typedef Documentation .....	291
19.2.6 Enumeration Type Documentation .....	292
19.2.7 Function Documentation .....	294
<b>19.3 LPUART DMA Driver .....</b>	<b>312</b>
19.3.1 Overview .....	312
19.3.2 Data Structure Documentation .....	313

Section No.	Title	Page No.
19.3.3	Macro Definition Documentation .....	313
19.3.4	Typedef Documentation .....	313
19.3.5	Function Documentation .....	314
<b>19.4</b>	<b>LPUART FreeRTOS Driver .....</b>	<b>318</b>
19.4.1	Overview .....	318
19.4.2	Data Structure Documentation .....	318
19.4.3	Macro Definition Documentation .....	319
19.4.4	Function Documentation .....	319
<b>Chapter 20 MMAU: Memory Mapped Arithmetic Unit</b>		
<b>20.1</b>	<b>Overview .....</b>	<b>322</b>
<b>20.2</b>	<b>Function groups .....</b>	<b>322</b>
20.2.1	MMAU Initialization .....	322
20.2.2	MMAU Interrupts .....	322
20.2.3	MMAU Instruction flags .....	322
20.2.4	MMAU Operators .....	322
<b>20.3</b>	<b>Typical use case and example .....</b>	<b>323</b>
<b>20.4</b>	<b>Enumeration Type Documentation .....</b>	<b>330</b>
20.4.1	mmau_interrupt_enable_t .....	330
20.4.2	mmau_interrupt_flag_t .....	331
20.4.3	mmau_instruction_flag_t .....	331
<b>20.5</b>	<b>Function Documentation .....</b>	<b>331</b>
20.5.1	MMAU_EnableDMA .....	331
20.5.2	MMAU_EnableSupervisorOnly .....	331
20.5.3	MMAU_Reset .....	332
20.5.4	MMAU_EnableInterrupts .....	332
20.5.5	MMAU_DisableInterrupts .....	332
20.5.6	MMAU_GetEnabledInterrupts .....	333
20.5.7	MMAU_GetInterruptFlags .....	333
20.5.8	MMAU_ClearInterruptFlags .....	334
20.5.9	MMAU_GetInstructionFlags .....	334
20.5.10	MMAU_SetInstructionFlags .....	334
20.5.11	MMAU_ClearInstructionFlags .....	335
20.5.12	MMAU_GetHwRevCmd .....	335
20.5.13	MMAU_ulta_d .....	335
20.5.14	MMAU_d_umul_ll .....	336
20.5.15	MMAU_d_umul_dl .....	336
20.5.16	MMAU_d_umuls_dl .....	337
20.5.17	MMAU_d_umula_l .....	338

<b>Section No.</b>	<b>Title</b>	<b>Page No.</b>
20.5.18	MMAU_d_umulas_l	338
20.5.19	MMAU_d_umac_ll	339
20.5.20	MMAU_d_umacs_ll	339
20.5.21	MMAU_d_umac_dl	340
20.5.22	MMAU_d_umacs_dl	340
20.5.23	MMAU_d_umaca_dl	341
20.5.24	MMAU_d_umacas_dl	341
20.5.25	MMAU_l_udiv_ll	342
20.5.26	MMAU_d_udiv_dl	342
20.5.27	MMAU_d_udiv_dd	343
20.5.28	MMAU_d_udiva_l	343
20.5.29	MMAU_d_udiva_d	344
20.5.30	MMAU_l_usqr_l	344
20.5.31	MMAU_l_usqr_d	345
20.5.32	MMAU_s_usqr_l	345
20.5.33	MMAU_l_usqra	346
20.5.34	MMAU_slda_d	346
20.5.35	MMAU_d_smul_ll	346
20.5.36	MMAU_d_smul_dl	347
20.5.37	MMAU_d_smuls_dl	347
20.5.38	MMAU_d_smula_l	348
20.5.39	MMAU_d_smulas_l	348
20.5.40	MMAU_d_smac_ll	349
20.5.41	MMAU_d_smacs_ll	349
20.5.42	MMAU_d_smac_dl	350
20.5.43	MMAU_d_smacs_dl	350
20.5.44	MMAU_d_smaca_dl	351
20.5.45	MMAU_d_smacas_dl	351
20.5.46	MMAU_l_sdiv_ll	352
20.5.47	MMAU_l_sdivs_ll	352
20.5.48	MMAU_d_sdiv_dl	353
20.5.49	MMAU_d_sdivs_dl	353
20.5.50	MMAU_d_sdiv_dd	354
20.5.51	MMAU_d_sdivs_dd	354
20.5.52	MMAU_d_sdive_l	355
20.5.53	MMAU_d_sdivas_l	355
20.5.54	MMAU_d_sdive_d	356
20.5.55	MMAU_d_sdivas_d	356
20.5.56	MMAU_lda_d	357
20.5.57	MMAU_l_mul_ll	357
20.5.58	MMAU_l_muls_ll	358
20.5.59	MMAU_d_mul_ll	359
20.5.60	MMAU_d_muls_ll	359
20.5.61	MMAU_d_mul_dl	360
20.5.62	MMAU_d_muls_dl	360

Section No.	Title	Page No.
20.5.63	MMAU_d_mula_1 .....	361
20.5.64	MMAU_d_mulas_1 .....	361
20.5.65	MMAU_l_mul_dl .....	362
20.5.66	MMAU_l_muls_dl .....	362
20.5.67	MMAU_l_mula_1 .....	363
20.5.68	MMAU_l_mulas_1 .....	363
20.5.69	MMAU_d_mac_ll .....	364
20.5.70	MMAU_d_macs_ll .....	364
20.5.71	MMAU_d_mac_dl .....	365
20.5.72	MMAU_d_macs_dl .....	365
20.5.73	MMAU_d_maca_dl .....	366
20.5.74	MMAU_d_macas_dl .....	366
20.5.75	MMAU_l_mac_ll .....	367
20.5.76	MMAU_l_macs_ll .....	367
20.5.77	MMAU_l_mac_dl .....	368
20.5.78	MMAU_l_macs_dl .....	368
20.5.79	MMAU_l_maca_dl .....	369
20.5.80	MMAU_l_macas_dl .....	369
20.5.81	MMAU_l_div_ll .....	370
20.5.82	MMAU_l_divs_ll .....	370
20.5.83	MMAU_l_divas_1 .....	371
20.5.84	MMAU_d_div_dl .....	371
20.5.85	MMAU_d_divs_dl .....	372
20.5.86	MMAU_d_diva_1 .....	372
20.5.87	MMAU_d_divas_1 .....	373
20.5.88	MMAU_l_diva_1 .....	373
20.5.89	MMAU_l_sqr_1 .....	374
20.5.90	MMAU_l_sqr_d .....	374
20.5.91	MMAU_l_sqra .....	375

## Chapter 21 PDB: Programmable Delay Block

<b>21.1</b>	<b>Overview .....</b>	<b>376</b>
<b>21.2</b>	<b>Typical use case .....</b>	<b>376</b>
21.2.1	Working as basic PDB counter with a PDB interrupt .....	376
21.2.2	Working with an additional trigger. The ADC trigger is used as an example .....	376
<b>21.3</b>	<b>Data Structure Documentation .....</b>	<b>380</b>
21.3.1	struct pdb_config_t .....	380
21.3.2	struct pdb_adc_pretrigger_config_t .....	381
21.3.3	struct pdb_dac_trigger_config_t .....	381
<b>21.4</b>	<b>Macro Definition Documentation .....</b>	<b>381</b>
21.4.1	FSL_PDB_DRIVER_VERSION .....	381

Section No.	Title	Page No.
<b>21.5 Enumeration Type Documentation</b>		<b>381</b>
21.5.1 <code>_pdb_status_flags</code>		382
21.5.2 <code>_pdb_adc_pretrigger_flags</code>		382
21.5.3 <code>_pdb_interrupt_enable</code>		382
21.5.4 <code>pdb_load_value_mode_t</code>		382
21.5.5 <code>pdb_prescaler_divider_t</code>		383
21.5.6 <code>pdb_divider_multiplication_factor_t</code>		383
21.5.7 <code>pdb_trigger_input_source_t</code>		383
21.5.8 <code>pdb_adc_trigger_channel_t</code>		384
21.5.9 <code>pdb_adc_pretrigger_t</code>		384
21.5.10 <code>pdb_dac_trigger_channel_t</code>		385
21.5.11 <code>pdb_pulse_out_trigger_channel_t</code>		385
21.5.12 <code>pdb_pulse_out_channel_mask_t</code>		385
<b>21.6 Function Documentation</b>		<b>385</b>
21.6.1 <code>PDB_Init</code>		385
21.6.2 <code>PDB_Deinit</code>		386
21.6.3 <code>PDB_GetDefaultConfig</code>		386
21.6.4 <code>PDB_Enable</code>		386
21.6.5 <code>PDB_DoSoftwareTrigger</code>		386
21.6.6 <code>PDB_DoLoadValues</code>		387
21.6.7 <code>PDB_EnableDMA</code>		387
21.6.8 <code>PDB_EnableInterrupts</code>		387
21.6.9 <code>PDB_DisableInterrupts</code>		387
21.6.10 <code>PDB_GetStatusFlags</code>		388
21.6.11 <code>PDB_ClearStatusFlags</code>		388
21.6.12 <code>PDB_SetModulusValue</code>		388
21.6.13 <code>PDB_GetCounterValue</code>		388
21.6.14 <code>PDB_SetCounterDelayValue</code>		389
21.6.15 <code>PDB_SetADCPreTriggerConfig</code>		389
21.6.16 <code>PDB_SetADCPreTriggerDelayValue</code>		389
21.6.17 <code>PDB_GetADCPreTriggerStatusFlags</code>		390
21.6.18 <code>PDB_ClearADCPreTriggerStatusFlags</code>		390
21.6.19 <code>PDB_EnablePulseOutTrigger</code>		390
21.6.20 <code>PDB_SetPulseOutTriggerDelayValue</code>		391

## Chapter 22 PIT: Periodic Interrupt Timer

<b>22.1 Overview</b>	<b>392</b>
<b>22.2 Function groups</b>	<b>392</b>
22.2.1 Initialization and deinitialization	392
22.2.2 Timer period Operations	392
22.2.3 Start and Stop timer operations	392
22.2.4 Status	393

Section No.	Title	Page No.
22.2.5	Interrupt .....	393
<b>22.3</b>	<b>Typical use case .....</b>	<b>393</b>
22.3.1	PIT tick example .....	393
<b>22.4</b>	<b>Data Structure Documentation .....</b>	<b>394</b>
22.4.1	struct pit_config_t .....	394
<b>22.5</b>	<b>Enumeration Type Documentation .....</b>	<b>395</b>
22.5.1	pit_chnl_t .....	395
22.5.2	pit_interrupt_enable_t .....	395
22.5.3	pit_status_flags_t .....	395
<b>22.6</b>	<b>Function Documentation .....</b>	<b>395</b>
22.6.1	PIT_Init .....	395
22.6.2	PIT_Deinit .....	396
22.6.3	PIT_GetDefaultConfig .....	396
22.6.4	PIT_SetTimerChainMode .....	396
22.6.5	PIT_EnableInterrupts .....	397
22.6.6	PIT_DisableInterrupts .....	397
22.6.7	PIT_GetEnabledInterrupts .....	397
22.6.8	PIT_GetStatusFlags .....	398
22.6.9	PIT_ClearStatusFlags .....	399
22.6.10	PIT_SetTimerPeriod .....	399
22.6.11	PIT_GetCurrentTimerCount .....	400
22.6.12	PIT_StartTimer .....	400
22.6.13	PIT_StopTimer .....	400
22.6.14	PIT_GetLifetimeTimerCount .....	401

## Chapter 23 PMC: Power Management Controller

<b>23.1</b>	<b>Overview .....</b>	<b>402</b>
<b>23.2</b>	<b>Data Structure Documentation .....</b>	<b>403</b>
23.2.1	struct pmc_low_volt_detect_config_t .....	403
23.2.2	struct pmc_low_volt_warning_config_t .....	403
23.2.3	struct pmc_bandgap_buffer_config_t .....	403
<b>23.3</b>	<b>Macro Definition Documentation .....</b>	<b>404</b>
23.3.1	FSL_PMC_DRIVER_VERSION .....	404
<b>23.4</b>	<b>Enumeration Type Documentation .....</b>	<b>404</b>
23.4.1	pmc_low_volt_detect_volt_select_t .....	404
23.4.2	pmc_low_volt_warning_volt_select_t .....	404
<b>23.5</b>	<b>Function Documentation .....</b>	<b>404</b>

Section No.	Title	Page No.
23.5.1	PMC_ConfigureLowVoltDetect .....	404
23.5.2	PMC_GetLowVoltDetectFlag .....	405
23.5.3	PMC_ClearLowVoltDetectFlag .....	405
23.5.4	PMC_ConfigureLowVoltWarning .....	405
23.5.5	PMC_GetLowVoltWarningFlag .....	406
23.5.6	PMC_ClearLowVoltWarningFlag .....	406
23.5.7	PMC_ConfigureBandgapBuffer .....	406
23.5.8	PMC_GetPeriphIOIsolationFlag .....	407
23.5.9	PMC_ClearPeriphIOIsolationFlag .....	407
23.5.10	PMC_IsRegulatorInRunRegulation .....	407

## Chapter 24 PORT: Port Control and Interrupts

<b>24.1</b>	<b>Overview .....</b>	<b>409</b>
<b>24.2</b>	<b>Data Structure Documentation .....</b>	<b>411</b>
24.2.1	struct port_digital_filter_config_t .....	411
24.2.2	struct port_pin_config_t .....	411
<b>24.3</b>	<b>Macro Definition Documentation .....</b>	<b>412</b>
24.3.1	FSL_PORT_DRIVER_VERSION .....	412
<b>24.4</b>	<b>Enumeration Type Documentation .....</b>	<b>412</b>
24.4.1	_port_pull .....	412
24.4.2	_port_slew_rate .....	412
24.4.3	_port_open_drain_enable .....	412
24.4.4	_port_lock_register .....	412
24.4.5	port_mux_t .....	412
24.4.6	port_interrupt_t .....	413
24.4.7	port_digital_filter_clock_source_t .....	413
<b>24.5</b>	<b>Function Documentation .....</b>	<b>413</b>
24.5.1	PORT_SetPinConfig .....	414
24.5.2	PORT_SetMultiplePinsConfig .....	414
24.5.3	PORT_SetPinMux .....	415
24.5.4	PORT_EnablePinsDigitalFilter .....	415
24.5.5	PORT_SetDigitalFilterConfig .....	416
24.5.6	PORT_SetPinInterruptConfig .....	416
24.5.7	PORT_GetPinsInterruptFlags .....	417
24.5.8	PORT_ClearPinsInterruptFlags .....	417

## Chapter 25 QTMR: Quad Timer Driver

<b>25.1</b>	<b>Overview .....</b>	<b>418</b>
-------------	-----------------------	------------

Section No.	Title	Page No.
<b>25.2 Data Structure Documentation</b>		<b>421</b>
25.2.1 struct qtmr_config_t		421
<b>25.3 Macro Definition Documentation</b>		<b>421</b>
25.3.1 FSL_QTMR_DRIVER_VERSION		421
<b>25.4 Enumeration Type Documentation</b>		<b>421</b>
25.4.1 qtmr_primary_count_source_t		422
25.4.2 qtmr_input_source_t		422
25.4.3 qtmr_counting_mode_t		422
25.4.4 qtmr_output_mode_t		423
25.4.5 qtmr_input_capture_edge_t		423
25.4.6 qtmr_reload_control_t		423
25.4.7 qtmr_debug_action_t		423
25.4.8 qtmr_interrupt_enable_t		424
25.4.9 qtmr_status_flags_t		424
<b>25.5 Function Documentation</b>		<b>424</b>
25.5.1 QTMR_Init		424
25.5.2 QTMR_Deinit		424
25.5.3 QTMR_GetDefaultConfig		425
25.5.4 QTMR_SetupPwm		425
25.5.5 QTMR_SetupInputCapture		426
25.5.6 QTMR_EnableInterrupts		427
25.5.7 QTMR_DisableInterrupts		427
25.5.8 QTMR_GetEnabledInterrupts		427
25.5.9 QTMR_GetStatus		428
25.5.10 QTMR_ClearStatusFlags		429
25.5.11 QTMR_SetTimerPeriod		429
25.5.12 QTMR_GetCurrentTimerCount		429
25.5.13 QTMR_StartTimer		430
25.5.14 QTMR_StopTimer		430

## Chapter 26 RCM: Reset Control Module Driver

<b>26.1 Overview</b>		<b>431</b>
<b>26.2 Data Structure Documentation</b>		<b>432</b>
26.2.1 struct rcm_reset_pin_filter_config_t		432
<b>26.3 Macro Definition Documentation</b>		<b>432</b>
26.3.1 FSL_RCM_DRIVER_VERSION		432
<b>26.4 Enumeration Type Documentation</b>		<b>432</b>
26.4.1 rcm_reset_source_t		432
26.4.2 rcm_run_wait_filter_mode_t		433

Section No.	Title	Page No.
<b>26.5 Function Documentation</b>		<b>433</b>
26.5.1 RCM_GetPreviousResetSources		433
26.5.2 RCM_GetStickyResetSources		433
26.5.3 RCM_ClearStickyResetSources		434
26.5.4 RCM_ConfigureResetPinFilter		434

## Chapter 27 RNGA: Random Number Generator Accelerator Driver

<b>27.1 Overview</b>		<b>436</b>
<b>27.2 RNGA Initialization</b>		<b>436</b>
<b>27.3 Get random data from RNGA</b>		<b>436</b>
<b>27.4 RNGA Set/Get Working Mode</b>		<b>436</b>
<b>27.5 Seed RNGA</b>		<b>436</b>
<b>27.6 Macro Definition Documentation</b>		<b>437</b>
27.6.1 FSL_RNGA_DRIVER_VERSION		437
<b>27.7 Enumeration Type Documentation</b>		<b>437</b>
27.7.1 <code>rnga_mode_t</code>		438
<b>27.8 Function Documentation</b>		<b>438</b>
27.8.1 RNGA_Init		438
27.8.2 RNGA_Deinit		438
27.8.3 RNGA_GetRandomData		438
27.8.4 RNGA_Seed		439
27.8.5 RNGA_SetMode		440
27.8.6 RNGA_GetMode		440

## Chapter 28 SIM: System Integration Module Driver

<b>28.1 Overview</b>		<b>441</b>
<b>28.2 Data Structure Documentation</b>		<b>441</b>
28.2.1 <code>struct sim_uid_t</code>		441
<b>28.3 Enumeration Type Documentation</b>		<b>442</b>
28.3.1 <code>_sim_flash_mode</code>		442
<b>28.4 Function Documentation</b>		<b>442</b>
28.4.1 SIM_GetUniqueId		442
28.4.2 SIM_SetFlashMode		442

Section No.	Title	Page No.
<b>Chapter 29 SLCD: Segment LCD Driver</b>		
<b>29.1</b>	<b>Overview</b>	<b>443</b>
<b>29.2</b>	<b>Plane Setting and Display Control</b>	<b>443</b>
<b>29.3</b>	<b>Typical use case</b>	<b>443</b>
29.3.1	SLCD Initialization operation	443
<b>29.4</b>	<b>Data Structure Documentation</b>	<b>447</b>
29.4.1	struct slcd_fault_detect_config_t	447
29.4.2	struct slcd_clock_config_t	448
29.4.3	struct slcd_config_t	448
<b>29.5</b>	<b>Macro Definition Documentation</b>	<b>450</b>
29.5.1	FSL_SLCD_DRIVER_VERSION	450
<b>29.6</b>	<b>Enumeration Type Documentation</b>	<b>450</b>
29.6.1	slcd_power_supply_option_t	450
29.6.2	slcd_regulated_voltage_trim_t	450
29.6.3	slcd_load_adjust_t	451
29.6.4	slcd_clock_src_t	451
29.6.5	slcd_alt_clock_div_t	451
29.6.6	slcd_clock_prescaler_t	452
29.6.7	slcd_duty_cycle_t	452
29.6.8	slcd_phase_type_t	452
29.6.9	slcd_phase_index_t	453
29.6.10	slcd_display_mode_t	453
29.6.11	slcd_blink_mode_t	453
29.6.12	slcd_blink_rate_t	453
29.6.13	slcd_fault_detect_clock_prescaler_t	454
29.6.14	slcd_fault_detect_sample_window_width_t	454
29.6.15	slcd_interrupt_enable_t	454
29.6.16	slcd_lowpower_behavior	454
<b>29.7</b>	<b>Function Documentation</b>	<b>455</b>
29.7.1	SLCD_Init	455
29.7.2	SLCD_Deinit	455
29.7.3	SLCD_GetDefaultConfig	455
29.7.4	SLCD_StartDisplay	455
29.7.5	SLCD_StopDisplay	456
29.7.6	SLCD_StartBlinkMode	456
29.7.7	SLCD_StopBlinkMode	456
29.7.8	SLCD_SetBackPlanePhase	456
29.7.9	SLCD_SetFrontPlaneSegments	457
29.7.10	SLCD_SetFrontPlaneOnePhase	457

Section No.	Title	Page No.
29.7.11	SLCD_GetFaultDetectCounter .....	458
29.7.12	SLCD_EnableInterrupts .....	458
29.7.13	SLCD_DisableInterrupts .....	459
29.7.14	SLCD_GetInterruptStatus .....	459
29.7.15	SLCD_ClearInterruptStatus .....	459

## Chapter 30 SMC: System Mode Controller Driver

<b>30.1</b>	<b>Overview .....</b>	<b>461</b>
<b>30.2</b>	<b>Typical use case .....</b>	<b>461</b>
30.2.1	Enter wait or stop modes .....	461
<b>30.3</b>	<b>Data Structure Documentation .....</b>	<b>463</b>
30.3.1	struct smc_power_mode_vlls_config_t .....	463
<b>30.4</b>	<b>Enumeration Type Documentation .....</b>	<b>463</b>
30.4.1	smc_power_mode_protection_t .....	464
30.4.2	smc_power_state_t .....	464
30.4.3	smc_run_mode_t .....	464
30.4.4	smc_stop_mode_t .....	464
30.4.5	smc_stop_submode_t .....	464
30.4.6	smc_partial_stop_option_t .....	465
30.4.7	anonymous enum .....	465
<b>30.5</b>	<b>Function Documentation .....</b>	<b>465</b>
30.5.1	SMC_SetPowerModeProtection .....	465
30.5.2	SMC_GetPowerModeState .....	465
30.5.3	SMC_PreEnterStopModes .....	466
30.5.4	SMC_PostExitStopModes .....	466
30.5.5	SMC_PreEnterWaitModes .....	466
30.5.6	SMC_PostExitWaitModes .....	466
30.5.7	SMC_SetPowerModeRun .....	466
30.5.8	SMC_SetPowerModeWait .....	466
30.5.9	SMC_SetPowerModeStop .....	467
30.5.10	SMC_SetPowerModeVlpr .....	467
30.5.11	SMC_SetPowerModeVlpw .....	467
30.5.12	SMC_SetPowerModeVlps .....	468
30.5.13	SMC_SetPowerModeVlls .....	468

## Chapter 31 SPI: Serial Peripheral Interface Driver

<b>31.1</b>	<b>Overview .....</b>	<b>469</b>
<b>31.2</b>	<b>SPI Driver .....</b>	<b>470</b>

Section No.	Title	Page No.
31.2.1	Overview .....	470
31.2.2	Typical use case .....	470
31.2.3	Data Structure Documentation .....	475
31.2.4	Macro Definition Documentation .....	477
31.2.5	Enumeration Type Documentation .....	477
31.2.6	Function Documentation .....	480
31.2.7	Variable Documentation .....	490
<b>31.3</b>	<b>SPI DMA Driver .....</b>	<b>491</b>
31.3.1	Overview .....	491
31.3.2	Data Structure Documentation .....	492
31.3.3	Macro Definition Documentation .....	492
31.3.4	Typedef Documentation .....	492
31.3.5	Function Documentation .....	492
<b>31.4</b>	<b>SPI FreeRTOS driver .....</b>	<b>496</b>
31.4.1	Overview .....	496
31.4.2	Macro Definition Documentation .....	496
31.4.3	Function Documentation .....	496
<b>31.5</b>	<b>SPI CMSIS driver .....</b>	<b>498</b>
31.5.1	Function groups .....	498
31.5.2	Typical use case .....	499

## Chapter 32 SYSPMU: System Memory Protection Unit

<b>32.1</b>	<b>Overview .....</b>	<b>500</b>
<b>32.2</b>	<b>Initialization and Deinitialization .....</b>	<b>500</b>
<b>32.3</b>	<b>Basic Control Operations .....</b>	<b>500</b>
<b>32.4</b>	<b>Data Structure Documentation .....</b>	<b>503</b>
32.4.1	struct sysmpu.hardware_info_t .....	503
32.4.2	struct sysmpu.access_err_info_t .....	504
32.4.3	struct sysmpu_rwxrights_master_access_control_t .....	504
32.4.4	struct sysmpu_rwrights_master_access_control_t .....	505
32.4.5	struct sysmpu_region_config_t .....	505
32.4.6	struct sysmpu_config_t .....	506
<b>32.5</b>	<b>Macro Definition Documentation .....</b>	<b>506</b>
32.5.1	FSL_SYSMPU_DRIVER_VERSION .....	507
32.5.2	SYSMPU_MASTER_RWATTRIBUTE_START_PORT .....	507
32.5.3	SYSMPU_REGION_RWXRIGHTS_MASTER_SHIFT .....	507
32.5.4	SYSMPU_REGION_RWXRIGHTS_MASTER_MASK .....	507
32.5.5	SYSMPU_REGION_RWXRIGHTS_MASTER_WIDTH .....	507

<b>Section No.</b>	<b>Title</b>	<b>Page No.</b>
32.5.6	SYSMPU_REGION_RWXRIGHTS_MASTER .....	507
32.5.7	SYSMPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT .....	507
32.5.8	SYSMPU_REGION_RWXRIGHTS_MASTER_PE_MASK .....	507
32.5.9	SYSMPU_REGION_RWXRIGHTS_MASTER_PE .....	507
32.5.10	SYSMPU_REGION_RWRIGHTS_MASTER_SHIFT .....	507
32.5.11	SYSMPU_REGION_RWRIGHTS_MASTER_MASK .....	507
32.5.12	SYSMPU_REGION_RWRIGHTS_MASTER .....	507
<b>32.6</b>	<b>Enumeration Type Documentation .....</b>	<b>508</b>
32.6.1	sysmpu_region_total_num_t .....	508
32.6.2	sysmpu_slave_t .....	508
32.6.3	sysmpu_err_access_control_t .....	508
32.6.4	sysmpu_err_access_type_t .....	508
32.6.5	sysmpu_err_attributes_t .....	508
32.6.6	sysmpu_supervisor_access_rights_t .....	509
32.6.7	sysmpu_user_access_rights_t .....	509
<b>32.7</b>	<b>Function Documentation .....</b>	<b>509</b>
32.7.1	SYSMPU_Init .....	509
32.7.2	SYSMPU_Deinit .....	509
32.7.3	SYSMPU_Enable .....	510
32.7.4	SYSMPU_RegionEnable .....	510
32.7.5	SYSMPU_GetHardwareInfo .....	510
32.7.6	SYSMPU_SetRegionConfig .....	511
32.7.7	SYSMPU_SetRegionAddr .....	511
32.7.8	SYSMPU_SetRegionRwxMasterAccessRights .....	511
32.7.9	SYSMPU_GetSlavePortErrorStatus .....	512
32.7.10	SYSMPU_GetDetailErrorAccessInfo .....	512
<b>Chapter 33 UART: Universal Asynchronous Receiver/Transmitter Driver</b>		
<b>33.1</b>	<b>Overview .....</b>	<b>514</b>
<b>33.2</b>	<b>UART Driver .....</b>	<b>515</b>
33.2.1	Overview .....	515
33.2.2	Typical use case .....	515
33.2.3	Data Structure Documentation .....	521
33.2.4	Macro Definition Documentation .....	523
33.2.5	Typedef Documentation .....	523
33.2.6	Enumeration Type Documentation .....	523
33.2.7	Function Documentation .....	525
33.2.8	Variable Documentation .....	541
<b>33.3</b>	<b>UART DMA Driver .....</b>	<b>542</b>
33.3.1	Overview .....	542

Section No.	Title	Page No.
33.3.2	Data Structure Documentation .....	543
33.3.3	Macro Definition Documentation .....	543
33.3.4	Typedef Documentation .....	543
33.3.5	Function Documentation .....	544
<b>33.4</b>	<b>UART FreeRTOS Driver .....</b>	<b>549</b>
33.4.1	Overview .....	549
33.4.2	Data Structure Documentation .....	549
33.4.3	Macro Definition Documentation .....	550
33.4.4	Function Documentation .....	550
<b>33.5</b>	<b>UART CMSIS Driver .....</b>	<b>552</b>
33.5.1	UART CMSIS Driver .....	552
<b>Chapter 34 VREF: Voltage Reference Driver</b>		
<b>34.1</b>	<b>Overview .....</b>	<b>554</b>
<b>34.2</b>	<b>VREF functional Operation .....</b>	<b>554</b>
<b>34.3</b>	<b>Typical use case and example .....</b>	<b>554</b>
<b>34.4</b>	<b>Data Structure Documentation .....</b>	<b>555</b>
34.4.1	struct vref_config_t .....	555
<b>34.5</b>	<b>Macro Definition Documentation .....</b>	<b>555</b>
34.5.1	FSL_VREF_DRIVER_VERSION .....	555
<b>34.6</b>	<b>Enumeration Type Documentation .....</b>	<b>555</b>
34.6.1	vref_buffer_mode_t .....	555
<b>34.7</b>	<b>Function Documentation .....</b>	<b>555</b>
34.7.1	VREF_Init .....	555
34.7.2	VREF_Deinit .....	556
34.7.3	VREF_GetDefaultConfig .....	556
34.7.4	VREF_SetTrimVal .....	557
34.7.5	VREF_GetTrimVal .....	557
34.7.6	VREF_SetLowReferenceTrimVal .....	557
34.7.7	VREF_GetLowReferenceTrimVal .....	558
<b>Chapter 35 WDOG: Watchdog Timer Driver</b>		
<b>35.1</b>	<b>Overview .....</b>	<b>559</b>
<b>35.2</b>	<b>Typical use case .....</b>	<b>559</b>

Section No.	Title	Page No.
<b>35.3 Data Structure Documentation</b>		<b>561</b>
35.3.1 struct wdog_work_mode_t		561
35.3.2 struct wdog_config_t		561
35.3.3 struct wdog_test_config_t		562
<b>35.4 Macro Definition Documentation</b>		<b>562</b>
35.4.1 FSL_WDOG_DRIVER_VERSION		562
<b>35.5 Enumeration Type Documentation</b>		<b>562</b>
35.5.1 wdog_clock_source_t		562
35.5.2 wdog_clock_prescaler_t		562
35.5.3 wdog_test_mode_t		563
35.5.4 wdog_tested_byte_t		563
35.5.5 _wdog_interrupt_enable_t		563
35.5.6 _wdog_status_flags_t		563
<b>35.6 Function Documentation</b>		<b>563</b>
35.6.1 WDOG_GetDefaultConfig		563
35.6.2 WDOG_Init		564
35.6.3 WDOG_Deinit		564
35.6.4 WDOG_SetTestModeConfig		565
35.6.5 WDOG_Enable		565
35.6.6 WDOG_Disable		565
35.6.7 WDOG_EnableInterrupts		566
35.6.8 WDOG_DisableInterrupts		566
35.6.9 WDOG_GetStatusFlags		566
35.6.10 WDOG_ClearStatusFlags		567
35.6.11 WDOG_SetTimeoutValue		567
35.6.12 WDOG_SetWindowValue		568
35.6.13 WDOG_Unlock		568
35.6.14 WDOG_Refresh		568
35.6.15 WDOG_GetResetCount		569
35.6.16 WDOG_ClearResetCount		570

## Chapter 36 XBAR: Inter-Peripheral Crossbar Switch

<b>36.1 Overview</b>	<b>571</b>
<b>36.2 Function groups</b>	<b>571</b>
36.2.1 XBAR Initialization	571
36.2.2 Call diagram	571
<b>36.3 Typical use case</b>	<b>571</b>
<b>36.4 Data Structure Documentation</b>	<b>572</b>
36.4.1 struct xbar_control_config_t	572

<b>Section No.</b>	<b>Title</b>	<b>Page No.</b>
<b>36.5 Enumeration Type Documentation</b>		<b>573</b>
36.5.1 xbar_active_edge_t		573
36.5.2 xbar_request_t		573
36.5.3 xbar_status_flag_t		573
<b>36.6 Function Documentation</b>		<b>574</b>
36.6.1 XBAR_Init		574
36.6.2 XBAR_Deinit		575
36.6.3 XBAR_SetSignalsConnection		575
36.6.4 XBAR_ClearStatusFlags		575
36.6.5 XBAR_GetStatusFlags		576
36.6.6 XBAR_SetOutputSignalConfig		576
<b>Chapter 37 Debug Console</b>		
<b>37.1 Overview</b>		<b>578</b>
<b>37.2 Function groups</b>		<b>578</b>
37.2.1 Initialization		578
37.2.2 Advanced Feature		579
37.2.3 SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART		583
<b>37.3 Typical use case</b>		<b>584</b>
<b>37.4 Macro Definition Documentation</b>		<b>586</b>
37.4.1 DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN		586
37.4.2 DEBUGCONSOLE_REDIRECT_TO_SDK		586
37.4.3 DEBUGCONSOLE_DISABLE		586
37.4.4 SDK_DEBUGCONSOLE		586
37.4.5 PRINTF		586
<b>37.5 Function Documentation</b>		<b>586</b>
37.5.1 DbgConsole_Init		586
37.5.2 DbgConsole_Deinit		587
37.5.3 DbgConsole_EnterLowpower		587
37.5.4 DbgConsole_ExitLowpower		588
37.5.5 DbgConsole_Printf		588
37.5.6 DbgConsole_Vprintf		588
37.5.7 DbgConsole_Putchar		588
37.5.8 DbgConsole_Scanf		589
37.5.9 DbgConsole_Getchar		589
37.5.10 DbgConsole_BlockingPrintf		590
37.5.11 DbgConsole_BlockingVprintf		590
37.5.12 DbgConsole_Flush		590
37.5.13 StrFormatPrintf		591

Section No.	Title	Page No.
37.5.14	StrFormatScanf .....	591
<b>37.6</b>	<b>Semihosting .....</b>	<b>592</b>
37.6.1	Guide Semihosting for IAR .....	592
37.6.2	Guide Semihosting for Keil µVision .....	592
37.6.3	Guide Semihosting for MCUXpresso IDE .....	593
37.6.4	Guide Semihosting for ARMGCC .....	593

## Chapter 38 Notification Framework

<b>38.1</b>	<b>Overview .....</b>	<b>596</b>
<b>38.2</b>	<b>Notifier Overview .....</b>	<b>596</b>
<b>38.3</b>	<b>Data Structure Documentation .....</b>	<b>598</b>
38.3.1	struct notifier_notification_block_t .....	598
38.3.2	struct notifier_callback_config_t .....	599
38.3.3	struct notifier_handle_t .....	599
<b>38.4</b>	<b>Typedef Documentation .....</b>	<b>600</b>
38.4.1	notifier_user_config_t .....	600
38.4.2	notifier_user_function_t .....	600
38.4.3	notifier_callback_t .....	601
<b>38.5</b>	<b>Enumeration Type Documentation .....</b>	<b>601</b>
38.5.1	_notifier_status .....	601
38.5.2	notifier_policy_t .....	602
38.5.3	notifier_notification_type_t .....	602
38.5.4	notifier_callback_type_t .....	602
<b>38.6</b>	<b>Function Documentation .....</b>	<b>602</b>
38.6.1	NOTIFIER_CreateHandle .....	603
38.6.2	NOTIFIER_SwitchConfig .....	604
38.6.3	NOTIFIER_GetErrorCallbackIndex .....	605

## Chapter 39 Shell

<b>39.1</b>	<b>Overview .....</b>	<b>606</b>
<b>39.2</b>	<b>Function groups .....</b>	<b>606</b>
39.2.1	Initialization .....	606
39.2.2	Advanced Feature .....	606
39.2.3	Shell Operation .....	606
<b>39.3</b>	<b>Data Structure Documentation .....</b>	<b>608</b>
39.3.1	struct shell_command_t .....	608

Section No.	Title	Page No.
<b>39.4 Macro Definition Documentation</b>		<b>609</b>
39.4.1 SHELL_NON_BLOCKING_MODE		609
39.4.2 SHELL_AUTO_COMPLETE		609
39.4.3 SHELL_BUFFER_SIZE		609
39.4.4 SHELL_MAX_ARGS		609
39.4.5 SHELL_HISTORY_COUNT		609
39.4.6 SHELL_HANDLE_SIZE		609
39.4.7 SHELL_USE_COMMON_TASK		609
39.4.8 SHELL_TASK_PRIORITY		609
39.4.9 SHELL_TASK_STACK_SIZE		609
39.4.10 SHELL_HANDLE_DEFINE		610
39.4.11 SHELL_COMMAND_DEFINE		610
39.4.12 SHELL_COMMAND		611
<b>39.5 Typedef Documentation</b>		<b>611</b>
39.5.1 cmd_function_t		611
<b>39.6 Enumeration Type Documentation</b>		<b>611</b>
39.6.1 shell_status_t		611
<b>39.7 Function Documentation</b>		<b>611</b>
39.7.1 SHELL_Init		611
39.7.2 SHELL_RegisterCommand		612
39.7.3 SHELL_UnregisterCommand		613
39.7.4 SHELL_Write		613
39.7.5 SHELL_Printf		613
39.7.6 SHELL_WriteSynchronization		614
39.7.7 SHELL_PrintfSynchronization		614
39.7.8 SHELL_ChangePrompt		615
39.7.9 SHELL_PrintPrompt		615
39.7.10 SHELL_Task		615
39.7.11 SHELL_checkRunningInIsr		616

## Chapter 40 Serial Manager

<b>40.1 Overview</b>		<b>617</b>
<b>40.2 Data Structure Documentation</b>		<b>620</b>
40.2.1 struct serial_manager_config_t		620
40.2.2 struct serial_manager_callback_message_t		620
<b>40.3 Macro Definition Documentation</b>		<b>621</b>
40.3.1 SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE		621
40.3.2 SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE		621
40.3.3 SERIAL_MANAGER_USE_COMMON_TASK		621

Section No.	Title	Page No.
40.3.4	SERIAL_MANAGER_HANDLE_SIZE .....	621
40.3.5	SERIAL_MANAGER_HANDLE_DEFINE .....	621
40.3.6	SERIAL_MANAGER_WRITE_HANDLE_DEFINE .....	621
40.3.7	SERIAL_MANAGER_READ_HANDLE_DEFINE .....	622
40.3.8	SERIAL_MANAGER_TASK_PRIORITY .....	622
40.3.9	SERIAL_MANAGER_TASK_STACK_SIZE .....	622
<b>40.4</b>	<b>Enumeration Type Documentation .....</b>	<b>622</b>
40.4.1	serial_port_type_t .....	622
40.4.2	serial_manager_type_t .....	623
40.4.3	serial_manager_status_t .....	623
<b>40.5</b>	<b>Function Documentation .....</b>	<b>623</b>
40.5.1	SerialManager_Init .....	623
40.5.2	SerialManager_Deinit .....	624
40.5.3	SerialManager_OpenWriteHandle .....	625
40.5.4	SerialManager_CloseWriteHandle .....	626
40.5.5	SerialManager_OpenReadHandle .....	626
40.5.6	SerialManager_CloseReadHandle .....	627
40.5.7	SerialManager_WriteBlocking .....	628
40.5.8	SerialManager_ReadBlocking .....	628
40.5.9	SerialManager_EnterLowpower .....	629
40.5.10	SerialManager_ExitLowpower .....	629
40.5.11	SerialManager_SetLowpowerCriticalCb .....	630
<b>40.6</b>	<b>Serial Port Uart .....</b>	<b>631</b>
40.6.1	Overview .....	631
40.6.2	Enumeration Type Documentation .....	631

# Chapter 1

## Introduction

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

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

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

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

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



<b>Deliverable</b>	<b>Location</b>
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](http://nxp.com)

Web Support: [nxp.com/support](http://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](http://nxp.com/SalesTermsandConditions).

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

© 2021 NXP B.V.

# Chapter 3

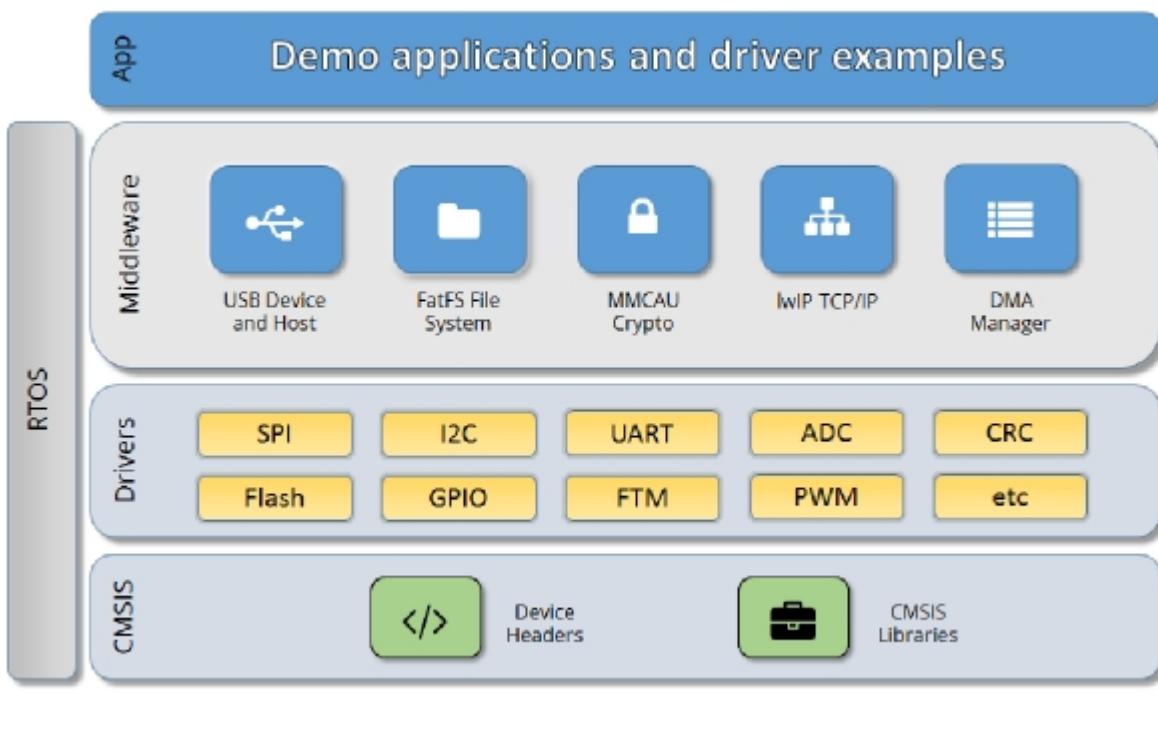
## Architectural Overview

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

### Overview

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

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



### MCU header files

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

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

## CMSIS Support

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

## MCUXpresso SDK Peripheral Drivers

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

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

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

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

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

## Interrupt handling for transactional APIs

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

```
PUBWEAK SPI0_IRQHandler  
PUBWEAK SPI0_DriverIRQHandler  
SPI0_IRQHandler
```

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

The first level of the weak implementation are the functions defined in the vector table. In the devices/<DEVICE\_NAME>/<TOOLCHAIN>/startup\_<DEVICE\_NAME>.s/.S file, the implementation of the first layer weak function calls the second layer of weak function. The implementation of the second layer weak function (ex. SPI0\_DriverIRQHandler) jumps to itself (B). The MCUXpresso SDK drivers with transactional APIs provide the reimplementation of the second layer function inside of the peripheral driver. If the MCUXpresso SDK drivers with transactional APIs are linked into the image, the SPI0\_DriverIRQHandler is replaced with the function implemented in the MCUXpresso SDK SPI driver.

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

The limitation of the double weak mechanism is that it cannot be used for peripherals that share the same vector entry. For this use case, redefine the first layer weak function to enable the desired peripheral interrupt functionality. For example, if the MCU's UART0 and UART1 share the same vector entry, redefine the UART0\_UART1\_IRQHandler according to the use case requirements.

## Feature Header Files

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

## Application

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

# Chapter 4

## Clock Driver

### 4.1 Overview

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

The clock driver supports:

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

### Modules

- Multipurpose Clock Generator (MCG)

### Files

- file `fsl_clock.h`

### Data Structures

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

### Macros

- `#define MCG_CONFIG_CHECK_PARAM 0U`  
*Configures whether to check a parameter in a function.*
- `#define FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL 0`  
*Configure whether driver controls clock.*
- `#define DMAMUX_CLOCKS`  
*Clock ip name array for DMAMUX.*
- `#define RTC_CLOCKS`  
*Clock ip name array for RTC.*
- `#define SPI_CLOCKS`  
*Clock ip name array for SPI.*
- `#define SLCD_CLOCKS`

- `#define EWM_CLOCKS`  
*Clock ip name array for EWM.*
- `#define AFE_CLOCKS`  
*Clock ip name array for AFE.*
- `#define LPUART_CLOCKS`  
*Clock ip name array for LPUART.*
- `#define ADC16_CLOCKS`  
*Clock ip name array for ADC16.*
- `#define XBAR_CLOCKS`  
*Clock ip name array for XBAR.*
- `#define SYSMPU_CLOCKS`  
*Clock ip name array for MPU.*
- `#define VREF_CLOCKS`  
*Clock ip name array for VREF.*
- `#define DMA_CLOCKS`  
*Clock ip name array for DMA.*
- `#define PORT_CLOCKS`  
*Clock ip name array for PORT.*
- `#define UART_CLOCKS`  
*Clock ip name array for UART.*
- `#define PIT_CLOCKS`  
*Clock ip name array for PIT.*
- `#define RNGA_CLOCKS`  
*Clock ip name array for RNGA.*
- `#define CRC_CLOCKS`  
*Clock ip name array for CRC.*
- `#define I2C_CLOCKS`  
*Clock ip name array for I2C.*
- `#define LPTMR_CLOCKS`  
*Clock ip name array for LPTMR.*
- `#define TMR_CLOCKS`  
*Clock ip name array for TMR.*
- `#define PDB_CLOCKS`  
*Clock ip name array for PDB.*
- `#define FTF_CLOCKS`  
*Clock ip name array for FTF.*
- `#define CMP_CLOCKS`  
*Clock ip name array for CMP.*
- `#define LPO_CLK_FREQ 1000U`  
*LPO clock frequency.*
- `#define SYS_CLK kCLOCK_CoreSysClk`  
*Peripherals clock source definition.*

## Enumerations

- enum `clock_name_t` {
   
kCLOCK\_CoreSysClk,
   
kCLOCK\_PlatClk,
   
kCLOCK\_BusClk,
   
kCLOCK\_FlashClk,
   
kCLOCK\_PllFllSelClk,
   
kCLOCK\_Er32kClk,
   
kCLOCK\_Osc0ErClk,
   
kCLOCK\_McgFixedFreqClk,
   
kCLOCK\_McgInternalRefClk,
   
kCLOCK\_McgFllClk,
   
kCLOCK\_McgPll0Clk,
   
kCLOCK\_McgExtPllClk,
   
kCLOCK\_McgPeriphClk,
   
kCLOCK\_LpoClk }

*Clock name used to get clock frequency.*

- enum `clock_ip_name_t`

*Clock gate name used for CLOCK\_EnableClock/CLOCK\_DisableClock.*

- enum `osc_mode_t` {
   
kOSC\_ModeExt = 0U,
   
kOSC\_ModeOscLowPower = MCG\_C2\_EREF0\_MASK,
   
kOSC\_ModeOscHighGain }

*OSC work mode.*

- enum `_osc_cap_load` {
   
kOSC\_Cap2P = OSC\_CR\_SC2P\_MASK,
   
kOSC\_Cap4P = OSC\_CR\_SC4P\_MASK,
   
kOSC\_Cap8P = OSC\_CR\_SC8P\_MASK,
   
kOSC\_Cap16P = OSC\_CR\_SC16P\_MASK }

*Oscillator capacitor load setting.*

- enum `_oscer_enable_mode` {
   
kOSC\_ErClkEnable = OSC\_CR\_ERCLKEN\_MASK,
   
kOSC\_ErClkEnableInStop = OSC\_CR\_EREFSTEN\_MASK }

*OSCERCLK enable mode.*

- enum `mcg_fll_src_t` {
   
kMCG\_FllSrcExternal,
   
kMCG\_FllSrcInternal }

*MCG FLL reference clock source select.*

- enum `mcg_irc_mode_t` {
   
kMCG\_IrcSlow,
   
kMCG\_IrcFast }

*MCG internal reference clock select.*

- enum `mcg_dmx32_t` {
   
kMCG\_Dmx32Default,
   
kMCG\_Dmx32Fine }

*MCG DCO Maximum Frequency with 32.768 kHz Reference.*

- enum `mcg_drs_t` {
   
  kMCG\_DrsLow,
   
  kMCG\_DrsMid,
   
  kMCG\_DrsMidHigh,
   
  kMCG\_DrsHigh }
   
    *MCG DCO range select.*
- enum `mcg_pll_ref_src_t` {
   
  kMCG\_PllRefRtc,
   
  kMCG\_PllRefIrc,
   
  kMCG\_PllRefFllRef }
   
    *MCG PLL reference clock select.*
- enum `mcg_clkout_src_t` {
   
  kMCG\_ClkOutSrcOut,
   
  kMCG\_ClkOutSrcInternal,
   
  kMCG\_ClkOutSrcExternal }
   
    *MCGOUT clock source.*
- enum `mcg_atm_select_t` {
   
  kMCG\_AtmSel32k,
   
  kMCG\_AtmSel4m }
   
    *MCG Automatic Trim Machine Select.*
- enum `mcg_oscsel_t` {
   
  kMCG\_OscselOsc,
   
  kMCG\_OscselRtc }
   
    *MCG OSC Clock Select.*
- enum `mcg_pll_clk_select_t` { kMCG\_PllClkSelPll0 }
   
    *MCG PLLCS select.*
- enum `mcg_monitor_mode_t` {
   
  kMCG\_MonitorNone,
   
  kMCG\_MonitorInt,
   
  kMCG\_MonitorReset }
   
    *MCG clock monitor mode.*
- enum `_mcg_status` {
   
  kStatus\_MCG\_ModeUnreachable = MAKE\_STATUS(kStatusGroup\_MCG, 0),
   
  kStatus\_MCG\_ModeInvalid = MAKE\_STATUS(kStatusGroup\_MCG, 1),
   
  kStatus\_MCG\_AtmBusClockInvalid = MAKE\_STATUS(kStatusGroup\_MCG, 2),
   
  kStatus\_MCG\_AtmDesiredFreqInvalid = MAKE\_STATUS(kStatusGroup\_MCG, 3),
   
  kStatus\_MCG\_AtmIrcUsed = MAKE\_STATUS(kStatusGroup\_MCG, 4),
   
  kStatus\_MCG\_AtmHardwareFail = MAKE\_STATUS(kStatusGroup\_MCG, 5),
   
  kStatus\_MCG\_SourceUsed = MAKE\_STATUS(kStatusGroup\_MCG, 6) }
   
    *MCG status.*
- enum `_mcg_status_flags_t` {
   
  kMCG\_Osc0LostFlag = (1U << 0U),
   
  kMCG\_Osc0InitFlag = (1U << 1U),
   
  kMCG\_RtcOscLostFlag = (1U << 4U),
   
  kMCG\_Pll0LostFlag = (1U << 5U),
   
  kMCG\_Pll0LockFlag = (1U << 6U) }
   
    *MCG status flags.*

- enum `_mcg_irclk_enable_mode` {
   
    `kMCG_IrclkEnable` = MCG\_C1\_IRCLKEN\_MASK,
   
    `kMCG_IrclkEnableInStop` = MCG\_C1\_IREFSTEN\_MASK }
   
    *MCG internal reference clock (MCGIRCLK) enable mode definition.*
- enum `_mcg_pll_enable_mode` {
   
    `kMCG_PllEnableIndependent` = MCG\_C5\_PLLCLKEN0\_MASK,
   
    `kMCG_PllEnableInStop` = MCG\_C5\_PLLSTEN0\_MASK }
   
    *MCG PLL clock enable mode definition.*
- enum `mcg_mode_t` {
   
    `kMCG_ModeFEI` = 0U,
   
    `kMCG_ModeFBI`,
   
    `kMCG_ModeBLPI`,
   
    `kMCG_ModeFEE`,
   
    `kMCG_ModeFBE`,
   
    `kMCG_ModeBLPE`,
   
    `kMCG_ModePBE`,
   
    `kMCG_ModePEE`,
   
    `kMCG_ModePEI`,
   
    `kMCG_ModePBI`,
   
    `kMCG_ModeError` }
   
    *MCG mode definitions.*

## Functions

- static void `CLOCK_EnableClock` (`clock_ip_name_t` name)
   
    *Enable the clock for specific IP.*
- static void `CLOCK_DisableClock` (`clock_ip_name_t` name)
   
    *Disable the clock for specific IP.*
- static void `CLOCK_SetEr32kClock` (`uint32_t` src)
   
    *Set ERCLK32K source.*
- static void `CLOCK_SetLpuartClock` (`uint32_t` src)
   
    *Set LPUART clock source.*
- static void `CLOCK_SetXbarClock` (`uint32_t` src)
   
    *Set XBAR clock source.*
- static void `CLOCK_SetAfeClkSrc` (`uint32_t` src)
   
    *Set the clock selection of AFECLKSEL.*
- static void `CLOCK_SetPllFllSelClock` (`uint32_t` src)
   
    *Set PLLFLLSEL clock source.*
- static void `CLOCK_SetClkOutClock` (`uint32_t` src)
   
    *Set CLKOUT source.*
- static void `CLOCK_SetAdcTriggerClock` (`uint32_t` src)
   
    *Set ADC trigger clock source.*
- static void `CLOCK_SetOutDiv` (`uint32_t` sysClk, `uint32_t` busClk, `uint32_t` flashClk)
   
    *System clock divider.*
- `uint32_t` `CLOCK_GetAfeFreq` (`void`)
   
    *Gets the clock frequency for AFE module.*
- `uint32_t` `CLOCK_GetFreq` (`clock_name_t` `clockName`)
   
    *Gets the clock frequency for a specific clock name.*
- `uint32_t` `CLOCK_GetCoreSysClkFreq` (`void`)

- Get the core clock or system clock frequency.
- `uint32_t CLOCK_GetPlatClkFreq (void)`  
Get the platform clock frequency.
- `uint32_t CLOCK_GetBusClkFreq (void)`  
Get the bus clock frequency.
- `uint32_t CLOCK_GetFlashClkFreq (void)`  
Get the flash clock frequency.
- `uint32_t CLOCK_GetPllFllSelClkFreq (void)`  
Get the output clock frequency selected by SIM[PLLFLLSEL].
- `uint32_t CLOCK_GetEr32kClkFreq (void)`  
Get the external reference 32K clock frequency (ERCLK32K).
- `void CLOCK_SetSimConfig (sim_clock_config_t const *config)`  
Set the clock configure in SIM module.
- `static void CLOCK_SetSimSafeDivs (void)`  
Set the system clock dividers in SIM to safe value.

## Variables

- `volatile uint32_t g_xtal0Freq`  
External XTAL0 (OSC0) clock frequency.
- `volatile uint32_t g_xtal32Freq`  
External XTAL32/EXTAL32/RTC\_CLKIN clock frequency.

## Driver version

- `#define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))`  
*CLOCK driver version 2.5.0.*

## MCG frequency functions.

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

## MCG clock configuration.

- `static void CLOCK_SetLowPowerEnable (bool enable)`  
Enables or disables the MCG low power.
- `status_t CLOCK_SetInternalRefClkConfig (uint8_t enableMode, mcg_irc_mode_t ircs, uint8_t fcdrv)`  
Configures the Internal Reference clock (MCGIRCLK).

- `status_t CLOCK_SetExternalRefClkConfig (mcg_oscsel_t oscsel)`  
*Selects the MCG external reference clock.*
- `static void CLOCK_SetFllExtRefDiv (uint8_t frdiv)`  
*Set the FLL external reference clock divider value.*
- `void CLOCK_EnablePll0 (mcg_pll_config_t const *config)`  
*Enables the PLL0 in FLL mode.*
- `static void CLOCK_DisablePll0 (void)`  
*Disables the PLL0 in FLL mode.*

## MCG clock lock monitor functions.

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

## OSC configuration

- `static void OSC_SetExtRefClkConfig (OSC_Type *base, oscer_config_t const *config)`  
*Configures the OSC external reference clock (OSCERCLK).*
- `static void OSC_SetCapLoad (OSC_Type *base, uint8_t capLoad)`  
*Sets the capacitor load configuration for the oscillator.*
- `void CLOCK_InitOsc0 (osc_config_t const *config)`  
*Initializes the OSC0.*
- `void CLOCK_DeinitOsc0 (void)`  
*Deinitializes the OSC0.*

## External clock frequency

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

## IRCs frequency

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

## MCG auto-trim machine.

- `status_t CLOCK_TrimInternalRefClk (uint32_t extFreq, uint32_t desireFreq, uint32_t *actualFreq, mcg_atm_select_t atms)`

*Auto trims the internal reference clock.*

## MCG mode functions.

- `mcg_mode_t CLOCK_GetMode (void)`  
*Gets the current MCG mode.*
- `status_t CLOCK_SetFeiMode (mcg_dmx32_t dmx32, mcg_drs_t drs, void(*fllStableDelay)(void))`  
*Sets the MCG to FEI mode.*
- `status_t CLOCK_SetFeeMode (uint8_t frdiv, mcg_dmx32_t dmx32, mcg_drs_t drs, void(*fllStableDelay)(void))`  
*Sets the MCG to FEE mode.*
- `status_t CLOCK_SetFbiMode (mcg_dmx32_t dmx32, mcg_drs_t drs, void(*fllStableDelay)(void))`  
*Sets the MCG to FBI mode.*
- `status_t CLOCK_SetFbeMode (uint8_t frdiv, mcg_dmx32_t dmx32, mcg_drs_t drs, void(*fllStableDelay)(void))`  
*Sets the MCG to FBE mode.*
- `status_t CLOCK_SetBlpiMode (void)`  
*Sets the MCG to BLPI mode.*
- `status_t CLOCK_SetBlpeMode (void)`  
*Sets the MCG to BLPE mode.*
- `status_t CLOCK_SetPbeMode (mcg_pll_clk_select_t pllcs, mcg_pll_config_t const *config)`  
*Sets the MCG to PBE mode.*
- `status_t CLOCK_SetPeeMode (void)`  
*Sets the MCG to PEE mode.*
- `status_t CLOCK_SetPbiMode (void)`  
*Sets the MCG to PBI mode.*
- `status_t CLOCK_SetPeiMode (void)`  
*Sets the MCG to PEI mode.*
- `status_t CLOCK_ExternalModeToFbeModeQuick (void)`  
*Switches the MCG to FBE mode from the external mode.*
- `status_t CLOCK_InternalModeToFbiModeQuick (void)`  
*Switches the MCG to FBI mode from internal modes.*
- `status_t CLOCK_BootToFeiMode (mcg_dmx32_t dmx32, mcg_drs_t drs, void(*fllStableDelay)(void))`  
*Sets the MCG to FEI mode during system boot up.*
- `status_t CLOCK_BootToFeeMode (mcg_oscsel_t oscsel, uint8_t frdiv, mcg_dmx32_t dmx32, mcg_drs_t drs, void(*fllStableDelay)(void))`  
*Sets the MCG to FEE mode during system bootup.*
- `status_t CLOCK_BootToBlpiMode (uint8_t fcrdiv, mcg_ircc_mode_t ircs, uint8_t ircEnableMode)`  
*Sets the MCG to BLPI mode during system boot up.*
- `status_t CLOCK_BootToBlpeMode (mcg_oscsel_t oscsel)`  
*Sets the MCG to BLPE mode during system boot up.*
- `status_t CLOCK_BootToPeeMode (mcg_oscsel_t oscsel, mcg_pll_clk_select_t pllcs, mcg_pll_config_t const *config)`  
*Sets the MCG to PEE mode during system boot up.*
- `status_t CLOCK_BootToPeiMode (void)`  
*Sets the MCG to PEI mode during system boot up.*
- `status_t CLOCK_SetMcgConfig (mcg_config_t const *config)`  
*Sets the MCG to a target mode.*

## 4.2 Data Structure Documentation

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

#### Data Fields

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

#### Field Documentation

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

### 4.2.2 struct oscer\_config\_t

#### Data Fields

- `uint8_t enableMode`  
*OSCERCLK enable mode.*

#### Field Documentation

- (1) `uint8_t oscer_config_t::enableMode`

OR'ed value of `_oscer_enable_mode`.

### 4.2.3 struct osc\_config\_t

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

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

#### Data Fields

- `uint32_t freq`  
*External clock frequency.*
- `uint8_t capLoad`

- *Capacitor load setting.*
- `osc_mode_t workMode`  
*OSC work mode setting.*
- `oscer_config_t oscerConfig`  
*Configuration for OSCERCLK.*

### Field Documentation

- (1) `uint32_t osc_config_t::freq`
- (2) `uint8_t osc_config_t::capLoad`
- (3) `osc_mode_t osc_config_t::workMode`
- (4) `oscer_config_t osc_config_t::oscerConfig`

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

#### Data Fields

- `uint8_t enableMode`  
*Enable mode.*
- `mcg_pll_ref_src_t refSrc`  
*PLL reference clock source.*
- `uint8_t frdiv`  
*FLL reference clock divider.*

### Field Documentation

- (1) `uint8_t mcg_pll_config_t::enableMode`  
OR'ed value of `_mcg_pll_enable_mode`.
- (2) `mcg_pll_ref_src_t mcg_pll_config_t::refSrc`
- (3) `uint8_t mcg_pll_config_t::frdiv`

### 4.2.5 struct mcg\_config\_t

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

1. frdiv: If the FLL uses the external reference clock, set this value to ensure that the external reference clock divided by frdiv is in the 31.25 kHz to 39.0625 kHz range.
2. The PLL reference clock divider PRDIV: PLL reference clock frequency after PRDIV should be in the `FSL_FEATURE_MCG_PLL_REF_MIN` to `FSL_FEATURE_MCG_PLL_REF_MAX` range.

## Data Fields

- `mcg_mode_t mcgMode`  
*MCG mode.*
- `uint8_t irclkEnableMode`  
*MCGIRCLK enable mode.*
- `mcg_irc_mode_t ircs`  
*Source, MCG\_C2[IRCS].*
- `uint8_t fcrdiv`  
*Divider, MCG\_SC[FCRDIV].*
- `uint8_t frdiv`  
*Divider MCG\_C1[FRDIV].*
- `mcg_drs_t drs`  
*DCO range MCG\_C4[DRST\_DRS].*
- `mcg_dmx32_t dmx32`  
*MCG\_C4[DMX32].*
- `mcg_oscsel_t oscsel`  
*OSC select MCG\_C7[OSCSEL].*
- `mcg_pll_config_t pll0Config`  
*MCGPLL0CLK configuration.*

## Field Documentation

- (1) `mcg_mode_t mcg_config_t::mcgMode`
- (2) `uint8_t mcg_config_t::irclkEnableMode`
- (3) `mcg_irc_mode_t mcg_config_t::ircs`
- (4) `uint8_t mcg_config_t::fcrdiv`
- (5) `uint8_t mcg_config_t::frdiv`
- (6) `mcg_drs_t mcg_config_t::drs`
- (7) `mcg_dmx32_t mcg_config_t::dmx32`
- (8) `mcg_oscsel_t mcg_config_t::oscsel`
- (9) `mcg_pll_config_t mcg_config_t::pll0Config`

## 4.3 Macro Definition Documentation

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

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

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

3. The users should only switch between the supported clock modes.

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

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

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

Note

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

#### 4.3.3 #define FSL\_CLOCK\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 0))

#### 4.3.4 #define DMAMUX\_CLOCKS

**Value:**

```
{
    \kCLOCK_Dmamux0 \
}
```

#### 4.3.5 #define RTC\_CLOCKS

**Value:**

```
{
    \kCLOCK_Rtc0 \
}
```

#### 4.3.6 #define SPI\_CLOCKS

**Value:**

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

#### 4.3.7 #define SLCD\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Slcd0 \
}
```

#### 4.3.8 #define EWM\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Ewm0 \
}
```

#### 4.3.9 #define AFE\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Afe0 \
}
```

#### 4.3.10 #define LPUART\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Lpuart0 \
}
```

#### 4.3.11 #define ADC16\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Adco \
}
```

#### 4.3.12 #define XBAR\_CLOCKS

**Value:**

```
{
    \_kCLOCK_Xbar \
}
```

#### 4.3.13 #define SY SMPU\_CLOCKS

**Value:**

```
{
    \_kCLOCK_Sysmpu0 \
}
```

#### 4.3.14 #define VREF\_CLOCKS

**Value:**

```
{
    \_kCLOCK_Vref0 \
}
```

#### 4.3.15 #define DMA\_CLOCKS

**Value:**

```
{
    \_kCLOCK_Dma0 \
}
```

#### 4.3.16 #define PORT\_CLOCKS

**Value:**

```
{
    \_kCLOCK_PortA, \_kCLOCK_PortB, \_kCLOCK_PortC, \_kCLOCK_PortD, \_kCLOCK_PortE, \_kCLOCK_PortF, \_kCLOCK_PortG, \
    \_kCLOCK_PortH, \_kCLOCK_PortI, \_kCLOCK_PortJ, \_kCLOCK_PortK, \_kCLOCK_PortL, \_kCLOCK_PortM \
}
```

### 4.3.17 #define UART\_CLOCKS

**Value:**

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

### 4.3.18 #define PIT\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Pit0, kCLOCK_Pit1 \
}
```

### 4.3.19 #define RNGA\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Rnga0 \
}
```

### 4.3.20 #define CRC\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Crc0 \
}
```

### 4.3.21 #define I2C\_CLOCKS

**Value:**

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

#### 4.3.22 #define LPTMR\_CLOCKS

**Value:**

```
{\n    kCLOCK_Lptmr0 \n}
```

#### 4.3.23 #define TMR\_CLOCKS

**Value:**

```
{\n    kCLOCK_Tmr0, kCLOCK_Tmr1, kCLOCK_Tmr2, kCLOCK_Tmr3 \n}
```

#### 4.3.24 #define PDB\_CLOCKS

**Value:**

```
{\n    kCLOCK_Pdb0 \n}
```

#### 4.3.25 #define FTF\_CLOCKS

**Value:**

```
{\n    kCLOCK_Ftf0 \n}
```

#### 4.3.26 #define CMP\_CLOCKS

**Value:**

```
{\n    kCLOCK_Cmp0, kCLOCK_Cmp1, kCLOCK_Cmp2 \n}
```

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

### 4.4 Enumeration Type Documentation

#### 4.4.1 enum clock\_name\_t

Enumerator

*kCLOCK\_CoreSysClk* Core/system clock.

*kCLOCK\_PlatClk* Platform clock.

*kCLOCK\_BusClk* Bus clock.

*kCLOCK\_FlashClk* Flash clock.

*kCLOCK\_PllFllSelClk* The clock after SIM[PLLFLSEL].

*kCLOCK\_Er32kClk* External reference 32K clock (ERCLK32K)

*kCLOCK\_Osc0ErClk* OSC0 external reference clock (OSC0ERCLK)

*kCLOCK\_McgFixedFreqClk* MCG fixed frequency clock (MCGFFCLK)

*kCLOCK\_McgInternalRefClk* MCG internal reference clock (MCGIRCLK)

*kCLOCK\_McgFllClk* MCGFLLCLK.

*kCLOCK\_McgPll0Clk* MCGPLL0CLK.

*kCLOCK\_McgExtPllClk* EXT\_PLLCLK.

*kCLOCK\_McgPeriphClk* MCG peripheral clock (MCGPCLK)

*kCLOCK\_LpoClk* LPO clock.

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

#### 4.4.3 enum osc\_mode\_t

Enumerator

*kOSC\_ModeExt* Use an external clock.

*kOSC\_ModeOscLowPower* Oscillator low power.

*kOSC\_ModeOscHighGain* Oscillator high gain.

#### 4.4.4 enum \_osc\_cap\_load

Enumerator

*kOSC\_Cap2P* 2 pF capacitor load

*kOSC\_Cap4P* 4 pF capacitor load

*kOSC\_Cap8P* 8 pF capacitor load

*kOSC\_Cap16P* 16 pF capacitor load

#### 4.4.5 enum \_oscer\_enable\_mode

Enumerator

*kOSC\_ErClkEnable* Enable.

*kOSC\_ErClkEnableInStop* Enable in stop mode.

#### 4.4.6 enum mcg\_fll\_src\_t

Enumerator

*kMCG\_FllSrcExternal* External reference clock is selected.

*kMCG\_FllSrcInternal* The slow internal reference clock is selected.

#### 4.4.7 enum mcg\_irc\_mode\_t

Enumerator

*kMCG\_IrcSlow* Slow internal reference clock selected.

*kMCG\_IrcFast* Fast internal reference clock selected.

#### 4.4.8 enum mcg\_dmx32\_t

Enumerator

*kMCG\_Dmx32Default* DCO has a default range of 25%.

*kMCG\_Dmx32Fine* DCO is fine-tuned for maximum frequency with 32.768 kHz reference.

#### 4.4.9 enum mcg\_drs\_t

Enumerator

*kMCG\_DrsLow* Low frequency range.

*kMCG\_DrsMid* Mid frequency range.

*kMCG\_DrsMidHigh* Mid-High frequency range.

*kMCG\_DrsHigh* High frequency range.

**4.4.10 enum mcg\_pll\_ref\_src\_t**

Enumerator

*kMCG\_PllRefRtc* Selects 32k RTC oscillator.*kMCG\_PllRefIrc* Selects 32k IRC.*kMCG\_PllRefFllRef* Selects FLL reference clock, the clock after FRDIV.**4.4.11 enum mcg\_clkout\_src\_t**

Enumerator

*kMCG\_ClkOutSrcOut* Output of the FLL is selected (reset default)*kMCG\_ClkOutSrcInternal* Internal reference clock is selected.*kMCG\_ClkOutSrcExternal* External reference clock is selected.**4.4.12 enum mcg\_atm\_select\_t**

Enumerator

*kMCG\_AtmSel32k* 32 kHz Internal Reference Clock selected*kMCG\_AtmSel4m* 4 MHz Internal Reference Clock selected**4.4.13 enum mcg\_oscsel\_t**

Enumerator

*kMCG\_OscselOsc* Selects System Oscillator (OSCCLK)*kMCG\_OscselRtc* Selects 32 kHz RTC Oscillator.**4.4.14 enum mcg\_pll\_clk\_select\_t**

Enumerator

*kMCG\_PllClkSelPll0* PLL0 output clock is selected.

#### 4.4.15 enum mcg\_monitor\_mode\_t

Enumerator

- kMCG\_MonitorNone*** Clock monitor is disabled.
- kMCG\_MonitorInt*** Trigger interrupt when clock lost.
- kMCG\_MonitorReset*** System reset when clock lost.

#### 4.4.16 enum \_mcg\_status

Enumerator

- kStatus\_MCG\_ModeUnreachable*** Can't switch to target mode.
- kStatus\_MCG\_ModeInvalid*** Current mode invalid for the specific function.
- kStatus\_MCG\_AtmBusClockInvalid*** Invalid bus clock for ATM.
- kStatus\_MCG\_AtmDesiredFreqInvalid*** Invalid desired frequency for ATM.
- kStatus\_MCG\_AtmIrcUsed*** IRC is used when using ATM.
- kStatus\_MCG\_AtmHardwareFail*** Hardware fail occurs during ATM.
- kStatus\_MCG\_SourceUsed*** Can't change the clock source because it is in use.

#### 4.4.17 enum \_mcg\_status\_flags\_t

Enumerator

- kMCG\_Osc0LostFlag*** OSC0 lost.
- kMCG\_Osc0InitFlag*** OSC0 crystal initialized.
- kMCG\_RtcOscLostFlag*** RTC OSC lost.
- kMCG\_Pl0LostFlag*** PLL0 lost.
- kMCG\_Pl0LockFlag*** PLL0 locked.

#### 4.4.18 enum \_mcg\_irclk\_enable\_mode

Enumerator

- kMCG\_IrclkEnable*** MCGIRCLK enable.
- kMCG\_IrclkEnableInStop*** MCGIRCLK enable in stop mode.

#### 4.4.19 enum \_mcg\_pll\_enable\_mode

Enumerator

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

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

***kMCG\_PlEnableInStop*** MCGPLLCLK enable in STOP mode.

#### 4.4.20 enum mcg\_mode\_t

Enumerator

***kMCG\_ModeFEI*** FEI - FLL Engaged Internal.

***kMCG\_ModeFBI*** FBI - FLL Bypassed Internal.

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

***kMCG\_ModeFEE*** FEE - FLL Engaged External.

***kMCG\_ModeFBE*** FBE - FLL Bypassed External.

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

***kMCG\_ModePBE*** PBE - PLL Bypassed External.

***kMCG\_ModePEE*** PEE - PLL Engaged External.

***kMCG\_ModePEI*** PEI - PLL Engaged Internal.

***kMCG\_ModePBI*** PBI - PLL Bypassed Internal.

***kMCG\_ModeError*** Unknown mode.

### 4.5 Function Documentation

#### 4.5.1 static void CLOCK\_EnableClock ( `clock_ip_name_t name` ) [inline], [static]

Parameters

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

#### 4.5.2 static void CLOCK\_DisableClock ( `clock_ip_name_t name` ) [inline], [static]

Parameters

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

#### 4.5.10 static void CLOCK\_SetOutDiv( uint32\_t *sysClk*, uint32\_t *busClk*, uint32\_t *flashClk* ) [inline], [static]

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

Parameters

<i>sysClk</i>	System clock divider value.
<i>busClk</i>	Bus clock divider value.
<i>flashClk</i>	Flash clock mode value.

#### 4.5.11 uint32\_t CLOCK\_GetAfeFreq( void )

This function checks the current mode configurations in MISC\_CTL register.

Returns

Clock frequency value in Hertz

#### 4.5.12 `uint32_t CLOCK_GetFreq( clock_name_t clockName )`

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

Parameters

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

Returns

Clock frequency value in Hertz

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

Returns

Clock frequency in Hz.

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

Returns

Clock frequency in Hz.

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

Returns

Clock frequency in Hz.

#### **4.5.16 uint32\_t CLOCK\_GetFlashClkFreq ( void )**

Returns

Clock frequency in Hz.

#### **4.5.17 uint32\_t CLOCK\_GetPIIFIISelClkFreq ( void )**

Returns

Clock frequency in Hz.

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

Returns

Clock frequency in Hz.

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

Returns

Clock frequency in Hz.

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

This function sets system layer clock settings in SIM module.

Parameters

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

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

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

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

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

Returns

The frequency of MCGOUTCLK.

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

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

Returns

The frequency of MCGFLLCLK.

#### **4.5.24 uint32\_t CLOCK\_GetInternalRefClkFreq ( void )**

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

Returns

The frequency of MCGIRCLK.

#### **4.5.25 uint32\_t CLOCK\_GetFixedFreqClkFreq ( void )**

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

Returns

The frequency of MCFFCLK.

#### **4.5.26 uint32\_t CLOCK\_GetPll0Freq ( void )**

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

Returns

The frequency of MCGPLL0CLK.

#### **4.5.27 static void CLOCK\_SetLowPowerEnable ( bool enable ) [inline], [static]**

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

Parameters

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

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

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

Parameters

<i>enableMode</i>	MCGIRCLK enable mode, OR'ed value of <a href="#">_mcg_irclk_enable_mode</a> .
<i>ircs</i>	MCGIRCLK clock source, choose fast or slow.
<i>fcrdiv</i>	Fast IRC divider setting (FCRDIV).

Return values

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

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

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

Parameters

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

Return values

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

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

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

Parameters

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

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

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

Parameters

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

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

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

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

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

Parameters

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

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

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

Parameters

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

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

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

Parameters

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

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

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

Example:

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

Returns

Logical OR value of the [\\_mcg\\_status\\_flags\\_t](#).

#### 4.5.37 void CLOCK\_ClearStatusFlags ( uint32\_t mask )

This function clears the MCG clock lock lost status. The parameter is a logical OR value of the flags to clear. See [\\_mcg\\_status\\_flags\\_t](#).

Example:

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

## Parameters

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

**4.5.38 static void OSC\_SetExtRefClkConfig ( OSC\_Type \* *base*, oscer\_config\_t const \* *config* ) [inline], [static]**

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

```
oscer_config_t config =
{
    .enableMode = kOSC_ErClkEnable |
        kOSC_ErClkEnableInStop,
    .erclkDiv   = 1U,
};

OSC_SetExtRefClkConfig(OSC, &config);
```

## Parameters

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

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

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

## Parameters

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

## Example:

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

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

This function initializes the OSC0 according to the board configuration.

Parameters

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

#### 4.5.41 void CLOCK\_DeinitOsc0 ( void )

This function deinitializes the OSC0.

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

Parameters

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

#### 4.5.43 static void CLOCK\_SetXtal32Freq ( uint32\_t *freq* ) [inline], [static]

Parameters

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

#### 4.5.44 void CLOCK\_SetSlowIrcFreq ( uint32\_t *freq* )

Parameters

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

#### 4.5.45 void CLOCK\_SetFastIrcFreq ( uint32\_t *freq* )

Parameters

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

#### 4.5.46 status\_t CLOCK\_TrimInternalRefClk ( uint32\_t extFreq, uint32\_t desireFreq, uint32\_t \* actualFreq, mcg\_atm\_select\_t atms )

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

Parameters

<code>extFreq</code>	External clock frequency, which should be a bus clock.
<code>desireFreq</code>	Frequency to trim to.
<code>actualFreq</code>	Actual frequency after trimming.
<code>atms</code>	Trim fast or slow internal reference clock.

Return values

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

#### 4.5.47 mcg\_mode\_t CLOCK\_GetMode ( void )

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

Returns

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

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

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

## Parameters

<i>dmx32</i>	DMX32 in FEI mode.
<i>drs</i>	The DCO range selection.
<i>fllStableDelay</i>	Delay function to ensure that the FLL is stable. Passing NULL does not cause a delay.

## Return values

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

## Note

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

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

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

## Parameters

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

## Return values

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

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

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

## Parameters

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

## Return values

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

## Note

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

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

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

## Parameters

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

## Return values

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

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

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

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

Return values

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

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

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

Return values

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

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

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

Parameters

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

Return values

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

Note

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

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

This function sets the MCG to PEE mode.

Return values

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

Note

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

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

This function sets the MCG to PBI mode.

Return values

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

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

This function sets the MCG to PEI mode.

Return values

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

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

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

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

Return values

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

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

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

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

Return values

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

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

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

## Parameters

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

## Return values

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

## Note

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

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

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

## Parameters

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

## Return values

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

#### 4.5.62 status\_t CLOCK\_BootToBLpiMode ( uint8\_t *fcrdiv*, mcg\_irc\_mode\_t *ircs*, uint8\_t *ircEnableMode* )

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

Parameters

<i>fcrdiv</i>	Fast IRC divider, FCRDIV.
<i>ircs</i>	The internal reference clock to select, IRCS.
<i>ircEnableMode</i>	The MCGIRCLK enable mode, OR'ed value of <a href="#">_mcg_irclk_enable_mode</a> .

Return values

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

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

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

Parameters

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

Return values

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

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

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

Parameters

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

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

Return values

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

#### 4.5.65 status\_t CLOCK\_BootToPeiMode ( void )

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

Return values

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

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

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

Parameters

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

Returns

Return *kStatus\_Success* if switched successfully; Otherwise, it returns an error code [\\_mcg\\_status](#).

Note

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

## 4.6 Variable Documentation

#### 4.6.1 volatile uint32\_t g\_xtal0Freq

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

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

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

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

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

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

## 4.7 Multipurpose Clock Generator (MCG)

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

### 4.7.1 Function description

MCG driver provides these functions:

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

#### 4.7.1.1 MCG frequency functions

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

#### 4.7.1.2 MCG clock configuration

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

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

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

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

#### 4.7.1.3 MCG clock lock monitor functions

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

#### 4.7.1.4 OSC configuration

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

#### 4.7.1.5 MCG auto-trim machine

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

#### 4.7.1.6 MCG mode functions

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

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

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

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

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

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

## 4.7.2 Typical use case

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

### 4.7.2.1 Switch between BLPI and FEI

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

### 4.7.2.2 Switch between BLPI and FEE

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

#### 4.7.2.3 Switch between BLPI and PEE

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

#### 4.7.2.4 Switch between BLPE and PEE

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

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

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

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

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

#### 4.7.2.5 Switch between BLPE and FEE

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

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

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

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

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

#### 4.7.2.6 Switch between BLPI and PEI

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

#### 4.7.3 Code Configuration Option

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

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

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

# Chapter 5

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

### 5.1 Overview

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

### 5.2 Typical use case

#### 5.2.1 Polling Configuration

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

#### 5.2.2 Interrupt Configuration

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

## Data Structures

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

## Enumerations

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

- enum `adc16_clock_divider_t` {
   
  `kADC16_ClockDivider1` = 0U,
   
  `kADC16_ClockDivider2` = 1U,
   
  `kADC16_ClockDivider4` = 2U,
   
  `kADC16_ClockDivider8` = 3U }
   
    *Clock divider for the converter.*
- enum `adc16_resolution_t` {
   
  `kADC16_Resolution8or9Bit` = 0U,
   
  `kADC16_Resolution12or13Bit` = 1U,
   
  `kADC16_Resolution10or11Bit` = 2U,
   
  `kADC16_ResolutionSE8Bit` = `kADC16_Resolution8or9Bit`,
   
  `kADC16_ResolutionSE12Bit` = `kADC16_Resolution12or13Bit`,
   
  `kADC16_ResolutionSE10Bit` = `kADC16_Resolution10or11Bit`,
   
  `kADC16_Resolution16Bit` = 3U,
   
  `kADC16_ResolutionSE16Bit` = `kADC16_Resolution16Bit` }
   
    *Converter's resolution.*
- enum `adc16_clock_source_t` {
   
  `kADC16_ClockSourceAlt0` = 0U,
   
  `kADC16_ClockSourceAlt1` = 1U,
   
  `kADC16_ClockSourceAlt2` = 2U,
   
  `kADC16_ClockSourceAlt3` = 3U,
   
  `kADC16_ClockSourceAsynchronousClock` = `kADC16_ClockSourceAlt3` }
   
    *Clock source.*
- enum `adc16_long_sample_mode_t` {
   
  `kADC16_LongSampleCycle24` = 0U,
   
  `kADC16_LongSampleCycle16` = 1U,
   
  `kADC16_LongSampleCycle10` = 2U,
   
  `kADC16_LongSampleCycle6` = 3U,
   
  `kADC16_LongSampleDisabled` = 4U }
   
    *Long sample mode.*
- enum `adc16_reference_voltage_source_t` {
   
  `kADC16_ReferenceVoltageSourceVref` = 0U,
   
  `kADC16_ReferenceVoltageSourceValt` = 1U,
   
  `kADC16_ReferenceVoltageSourceBandgap` = 2U }
   
    *Reference voltage source.*
- enum `adc16_hardware_average_mode_t` {
   
  `kADC16_HardwareAverageCount4` = 0U,
   
  `kADC16_HardwareAverageCount8` = 1U,
   
  `kADC16_HardwareAverageCount16` = 2U,
   
  `kADC16_HardwareAverageCount32` = 3U,
   
  `kADC16_HardwareAverageDisabled` = 4U }
   
    *Hardware average mode.*
- enum `adc16_hardware_compare_mode_t` {
   
  `kADC16_HardwareCompareMode0` = 0U,
   
  `kADC16_HardwareCompareMode1` = 1U,
   
  `kADC16_HardwareCompareMode2` = 2U,

```
kADC16_HardwareCompareMode3 = 3U }

Hardware compare mode.
```

## Driver version

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

## Initialization

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

## Advanced Features

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

## Conversion Channel

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

## 5.3 Data Structure Documentation

### 5.3.1 struct adc16\_config\_t

#### Data Fields

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

#### Field Documentation

- (1) `adc16_reference_voltage_source_t adc16_config_t::referenceVoltageSource`
- (2) `adc16_clock_source_t adc16_config_t::clockSource`
- (3) `bool adc16_config_t::enableAsynchronousClock`
- (4) `adc16_clock_divider_t adc16_config_t::clockDivider`
- (5) `adc16_resolution_t adc16_config_t::resolution`
- (6) `adc16_long_sample_mode_t adc16_config_t::longSampleMode`
- (7) `bool adc16_config_t::enableHighSpeed`
- (8) `bool adc16_config_t::enableLowPower`
- (9) `bool adc16_config_t::enableContinuousConversion`
- (10) `adc16_hardware_average_mode_t adc16_config_t::hardwareAverageMode`

### 5.3.2 struct adc16\_hardware\_compare\_config\_t

#### Data Fields

- [adc16\\_hardware\\_compare\\_mode\\_t hardwareCompareMode](#)  
*Select the hardware compare mode.*
- [int16\\_t value1](#)  
*Setting value1 for hardware compare mode.*
- [int16\\_t value2](#)  
*Setting value2 for hardware compare mode.*

#### Field Documentation

(1) [adc16\\_hardware\\_compare\\_mode\\_t adc16\\_hardware\\_compare\\_config\\_t::hardwareCompareMode](#)

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

(2) [int16\\_t adc16\\_hardware\\_compare\\_config\\_t::value1](#)

(3) [int16\\_t adc16\\_hardware\\_compare\\_config\\_t::value2](#)

### 5.3.3 struct adc16\_channel\_config\_t

#### Data Fields

- [uint32\\_t channelNumber](#)  
*Setting the conversion channel number.*
- [bool enableInterruptOnConversionCompleted](#)  
*Generate an interrupt request once the conversion is completed.*

#### Field Documentation

(1) [uint32\\_t adc16\\_channel\\_config\\_t::channelNumber](#)

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

(2) [bool adc16\\_channel\\_config\\_t::enableInterruptOnConversionCompleted](#)

## 5.4 Macro Definition Documentation

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

## 5.5 Enumeration Type Documentation

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

Enumerator

*kADC16\_ChannelConversionDoneFlag* Conversion done.

### 5.5.2 enum \_adc16\_status\_flags

Enumerator

*kADC16\_ActiveFlag* Converter is active.

*kADC16\_CalibrationFailedFlag* Calibration is failed.

### 5.5.3 enum adc16\_channel\_mux\_mode\_t

For some ADC16 channels, there are two pin selections in channel multiplexer. For example, ADC0\_SE4a and ADC0\_SE4b are the different channels that share the same channel number.

Enumerator

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

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

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

Enumerator

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

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

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

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

### 5.5.5 enum adc16\_resolution\_t

Enumerator

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

*kADC16\_Resolution12or13Bit* Single End 12-bit or Differential Sample 13-bit.

*kADC16\_Resolution10or11Bit* Single End 10-bit or Differential Sample 11-bit.

*kADC16\_ResolutionSE8Bit* Single End 8-bit.

*kADC16\_ResolutionSE12Bit* Single End 12-bit.

*kADC16\_ResolutionSE10Bit* Single End 10-bit.

*kADC16\_Resolution16Bit* Single End 16-bit or Differential Sample 16-bit.

*kADC16\_ResolutionSE16Bit* Single End 16-bit.

## 5.5.6 enum adc16\_clock\_source\_t

Enumerator

*kADC16\_ClockSourceAlt0* Selection 0 of the clock source.

*kADC16\_ClockSourceAlt1* Selection 1 of the clock source.

*kADC16\_ClockSourceAlt2* Selection 2 of the clock source.

*kADC16\_ClockSourceAlt3* Selection 3 of the clock source.

*kADC16\_ClockSourceAsynchronousClock* Using internal asynchronous clock.

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

Enumerator

*kADC16\_LongSampleCycle24* 20 extra ADCK cycles, 24 ADCK cycles total.

*kADC16\_LongSampleCycle16* 12 extra ADCK cycles, 16 ADCK cycles total.

*kADC16\_LongSampleCycle10* 6 extra ADCK cycles, 10 ADCK cycles total.

*kADC16\_LongSampleCycle6* 2 extra ADCK cycles, 6 ADCK cycles total.

*kADC16\_LongSampleDisabled* Disable the long sample feature.

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

Enumerator

*kADC16\_ReferenceVoltageSourceVref* For external pins pair of VrefH and VrefL.

*kADC16\_ReferenceVoltageSourceValt* For alternate reference pair of ValtH and ValtL.

*kADC16\_ReferenceVoltageSourceBandgap* For bandgap voltage from PMC.

## 5.5.9 enum adc16\_hardware\_average\_mode\_t

Enumerator

*kADC16\_HardwareAverageCount4* For hardware average with 4 samples.

*kADC16\_HardwareAverageCount8* For hardware average with 8 samples.

*kADC16\_HardwareAverageCount16* For hardware average with 16 samples.

*kADC16\_HardwareAverageCount32* For hardware average with 32 samples.

*kADC16\_HardwareAverageDisabled* Disable the hardware average feature.

### 5.5.10 enum adc16\_hardware\_compare\_mode\_t

Enumerator

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

## 5.6 Function Documentation

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

Parameters

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

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

Parameters

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

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

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

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

Parameters

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

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

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

Parameters

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

Returns

Execution status.

Return values

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

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

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

Parameters

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

### 5.6.9 void ADC16\_SetHardwareCompareConfig ( ADC\_Type \* *base*, const adc16\_hardware\_compare\_config\_t \* *config* )

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

Parameters

<i>base</i>	ADC16 peripheral base address.
<i>config</i>	Pointer to the "adc16_hardware_compare_config_t" structure. Passing "NULL" disables the feature.

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

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

Parameters

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

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

Parameters

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

Returns

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

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

Parameters

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

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

## Parameters

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

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

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

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

## Parameters

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

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

Parameters

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

Returns

Conversion value.

### 5.6.16 `uint32_t ADC16_GetChannelStatusFlags ( ADC_Type * base, uint32_t channelGroup )`

Parameters

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

Returns

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

# Chapter 6

## AFE: Analog Front End Driver

### 6.1 Overview

The MCUXpresso SDK provides a driver for the Analog Front End (AFE) module of MCUXpresso SDK devices.

The Analog Front End or AFE is an integrated module that is comprised of ADCs, PGA, filtering, and phase compensation blocks. The AFE is responsible for measuring the phase voltage, phase current, and neutral current.

### 6.2 Function groups

#### 6.2.1 Channel configuration structures

The driver uses instances of the channel configuration structures to configuration and initialization AFE channel. This structure holds the settings of the AFE measurement channel. The settings include AFE hardware/software triggering, AFE continuous/Single conversion mode, AFE channel mode, AFE channel analog gain, AFE channel oversampling ration. The AFE channel mode selects whether the bypass mode is enabled or disabled and the external clock selection.

#### 6.2.2 User configuration structures

The AFE driver uses instances of the user configuration structure `afe_config_t` for the AFE driver configuration. This structure holds the configuration which is common for all AFE channels. The settings include AFE low-power mode, AFE result format, AFE clock divider mode, AFE clock source mode, and AFE start up delay of modulators.

#### 6.2.3 AFE Initialization

To initialize the AFE driver for a typical use case, call the `AFE_GetDefaultConfig()` function which populates the structure. Then, call the `AFE_Init()` function and pass the base address of the AFE peripheral and a pointer to the user configuration structure.

To configure the AFE channel, for a typical use case call the `AFE_GetDefaultChnConfig()` function which populates the structure. Then, call the `AFE_SetChnConfig()` function and pass the base address of the AFE peripheral and a pointer to the channel configuration structure.

## 6.2.4 AFE Conversion

The driver contains functions for software triggering, a channel delay after trigger setting, a result (raw or converted to right justified), reading, and waiting functions.

If the software triggering is enabled (hwTriggerEnable parameter in afe\_chn\_config\_t is a false value), call the AFE\_SoftTriggerConv() function to start conversion.

## 6.3 Typical use case

### 6.3.1 AFE Initialization

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

### 6.3.2 AFE Conversion

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

## Data Structures

- struct `afe_channel_config_t`  
*Defines the structure to initialize the AFE channel. [More...](#)*
- struct `afe_config_t`  
*Defines the structure to initialize the AFE module. [More...](#)*

## Enumerations

- enum `_afe_channel_status_flag` {
   
  `kAFE_Channel0OverflowFlag` = AFE\_SR\_OVR0\_MASK,  
`kAFE_Channel1OverflowFlag` = AFE\_SR\_OVR1\_MASK,  
`kAFE_Channel2OverflowFlag` = AFE\_SR\_OVR2\_MASK,  
`kAFE_Channel0ReadyFlag` = AFE\_SR\_RDY0\_MASK,  
`kAFE_Channel1ReadyFlag` = AFE\_SR\_RDY1\_MASK,  
`kAFE_Channel2ReadyFlag` = AFE\_SR\_RDY2\_MASK,  
`kAFE_Channel0ConversionCompleteFlag` = AFE\_SR\_COCE0\_MASK,  
`kAFE_Channel1ConversionCompleteFlag` = AFE\_SR\_COCE1\_MASK,  
`kAFE_Channel2ConversionCompleteFlag` = AFE\_SR\_COCE2\_MASK,  
`kAFE_Channel3OverflowFlag` = AFE\_SR\_OVR3\_MASK,  
`kAFE_Channel3ReadyFlag` = AFE\_SR\_RDY3\_MASK,  
`kAFE_Channel3ConversionCompleteFlag` = AFE\_SR\_COCE3\_MASK }
   
*Defines the type of status flags.*
- enum {
   
  `kAFE_Channel0InterruptEnable` = AFE\_DI\_INTEN0\_MASK,  
`kAFE_Channel1InterruptEnable` = AFE\_DI\_INTEN1\_MASK,  
`kAFE_Channel2InterruptEnable` = AFE\_DI\_INTEN2\_MASK,  
`kAFE_Channel3InterruptEnable` = AFE\_DI\_INTEN3\_MASK }

- Defines AFE interrupt enable.
- enum {
   
kAFE\_Channel0DMAEnable = AFE\_DI\_DMAEN0\_MASK,  
 kAFE\_Channel1DMAEnable = AFE\_DI\_DMAEN1\_MASK,  
 kAFE\_Channel2DMAEnable = AFE\_DI\_DMAEN2\_MASK,  
 kAFE\_Channel3DMAEnable = AFE\_DI\_DMAEN3\_MASK }
- Defines AFE DMA enable.
- enum {
   
kAFE\_Channel0Trigger = AFE\_CR\_SOFT\_TRG0\_MASK,  
 kAFE\_Channel1Trigger = AFE\_CR\_SOFT\_TRG1\_MASK,  
 kAFE\_Channel2Trigger = AFE\_CR\_SOFT\_TRG2\_MASK,  
 kAFE\_Channel3Trigger = AFE\_CR\_SOFT\_TRG3\_MASK }
- Defines AFE channel trigger flag.
- enum afe\_decimator\_oversample\_ratio\_t {
   
kAFE\_DecimatorOversampleRatio64 = 0U,  
 kAFE\_DecimatorOversampleRatio128 = 1U,  
 kAFE\_DecimatorOversampleRatio256 = 2U,  
 kAFE\_DecimatorOversampleRatio512 = 3U,  
 kAFE\_DecimatorOversampleRatio1024 = 4U,  
 kAFE\_DecimatorOversampleRatio2048 = 5U }
- AFE OSR modes.
- enum afe\_result\_format\_t {
   
kAFE\_ResultFormatLeft = 0U,  
 kAFE\_ResultFormatRight = 1U }
- Defines the AFE result format modes.
- enum afe\_clock\_divider\_t {
   
kAFE\_ClockDivider1 = 0U,  
 kAFE\_ClockDivider2 = 1U,  
 kAFE\_ClockDivider4 = 2U,  
 kAFE\_ClockDivider8 = 3U,  
 kAFE\_ClockDivider16 = 4U,  
 kAFE\_ClockDivider32 = 5U,  
 kAFE\_ClockDivider64 = 6U,  
 kAFE\_ClockDivider128 = 7U,  
 kAFE\_ClockDivider256 = 8U }
- Defines the AFE clock divider modes.
- enum afe\_clock\_source\_t {
   
kAFE\_ClockSource0 = 0U,  
 kAFE\_ClockSource1 = 1U,  
 kAFE\_ClockSource2 = 2U,  
 kAFE\_ClockSource3 = 3U }
- Defines the AFE clock source modes.
- enum afe\_pga\_gain\_t {

```
kAFE_PgaDisable = 0U,
kAFE_PgaGain1 = 1U,
kAFE_PgaGain2 = 2U,
kAFE_PgaGain4 = 3U,
kAFE_PgaGain8 = 4U,
kAFE_PgaGain16 = 5U,
kAFE_PgaGain32 = 6U }
```

*Defines the PGA's values.*

- enum `afe_bypass_mode_t` {
 kAFE\_BypassInternalClockPositiveEdge = 0U,
 kAFE\_BypassExternalClockPositiveEdge = 1U,
 kAFE\_BypassInternalClockNegativeEdge = 2U,
 kAFE\_BypassExternalClockNegativeEdge = 3U,
 kAFE\_BypassDisable = 4U }

*Defines the bypass modes.*

## Driver version

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

## AFE Initialization

- void `AFE_Init` (AFE\_Type \*base, const `afe_config_t` \*config)  
*Initialization for the AFE module.*
- void `AFE_Deinit` (AFE\_Type \*base)  
*De-Initialization for the AFE module.*
- void `AFE_GetDefaultConfig` (`afe_config_t` \*config)  
*Fills the user configure structure.*
- static void `AFE_SoftwareReset` (AFE\_Type \*base, bool enable)  
*Software reset the AFE module.*
- static void `AFE_Enable` (AFE\_Type \*base, bool enable)  
*Enables all configured AFE channels.*

## AFE Conversion

- void `AFE_SetChannelConfig` (AFE\_Type \*base, uint32\_t channel, const `afe_channel_config_t` \*config)  
*Configure the selected AFE channel.*
- void `AFE_GetDefaultChannelConfig` (`afe_channel_config_t` \*config)  
*Fills the channel configuration structure.*
- uint32\_t `AFE_GetChannelConversionValue` (AFE\_Type \*base, uint32\_t channel)  
*Reads the raw conversion value.*
- static void `AFE_DoSoftwareTriggerChannel` (AFE\_Type \*base, uint32\_t mask)  
*Triggers the AFE conversion by software.*
- static uint32\_t `AFE_GetChannelStatusFlags` (AFE\_Type \*base)  
*Gets the AFE status flag state.*
- void `AFE_SetChannelPhaseDelayValue` (AFE\_Type \*base, uint32\_t channel, uint32\_t value)  
*Sets phase delays value.*

- static void [AFE\\_SetChannelPhasetDelayOk](#) (AFE\_Type \*base)  
*Asserts the phase delay setting.*
- static void [AFE\\_EnableChannelInterrupts](#) (AFE\_Type \*base, uint32\_t mask)  
*Enables AFE interrupt.*
- static void [AFE\\_DisableChannelInterrupts](#) (AFE\_Type \*base, uint32\_t mask)  
*Disables AFE interrupt.*
- static uint32\_t [AFE\\_GetEnabledChannelInterrupts](#) (AFE\_Type \*base)  
*Returns mask of all enabled AFE interrupts.*
- void [AFE\\_EnableChannelDMA](#) (AFE\_Type \*base, uint32\_t mask, bool enable)  
*Enables/Disables AFE DMA.*

## 6.4 Data Structure Documentation

### 6.4.1 struct afe\_channel\_config\_t

This structure keeps the configuration for the AFE channel.

#### Data Fields

- bool [enableHardwareTrigger](#)  
*Enable triggering by hardware.*
- bool [enableContinuousConversion](#)  
*Enable continuous conversion mode.*
- [afe\\_bypass\\_mode\\_t](#) [channelMode](#)  
*Select if channel is in bypassed mode.*
- [afe\\_pga\\_gain\\_t](#) [pgaGainSelect](#)  
*Select the analog gain applied to the input signal.*
- [afe\\_decimator\\_oversample\\_ratio\\_t](#) [decimatorOversampleRatio](#)  
*Select the over sampling ration.*

#### Field Documentation

- (1) **bool afe\_channel\_config\_t::enableHardwareTrigger**
- (2) **bool afe\_channel\_config\_t::enableContinuousConversion**
- (3) **afe\_bypass\_mode\_t afe\_channel\_config\_t::channelMode**
- (4) **afe\_pga\_gain\_t afe\_channel\_config\_t::pgaGainSelect**
- (5) **afe\_decimator\_oversample\_ratio\_t afe\_channel\_config\_t::decimatorOversampleRatio**

### 6.4.2 struct afe\_config\_t

This structure keeps the configuration for the AFE module.

## Data Fields

- bool `enableLowPower`  
*Enable low power mode.*
- `afe_result_format_t resultFormat`  
*Select the result format.*
- `afe_clock_divider_t clockDivider`  
*Select the clock divider ration for the modulator clock.*
- `afe_clock_source_t clockSource`  
*Select clock source for modulator clock.*
- `uint8_t startupCount`  
*Select the start up delay of modulators.*

## Field Documentation

- (1) `bool afe_config_t::enableLowPower`
- (2) `afe_result_format_t afe_config_t::resultFormat`
- (3) `afe_clock_divider_t afe_config_t::clockDivider`
- (4) `afe_clock_source_t afe_config_t::clockSource`
- (5) `uint8_t afe_config_t::startupCount`

## 6.5 Macro Definition Documentation

### 6.5.1 `#define FSL_AFE_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

## 6.6 Enumeration Type Documentation

### 6.6.1 `enum _afe_channel_status_flag`

Enumerator

**`kAFE_Channel0OverflowFlag`** Channel 0 previous conversion result has not been read and new data has already arrived.

**`kAFE_Channel1OverflowFlag`** Channel 1 previous conversion result has not been read and new data has already arrived.

**`kAFE_Channel2OverflowFlag`** Channel 2 previous conversion result has not been read and new data has already arrived.

**`kAFE_Channel0ReadyFlag`** Channel 0 is ready to conversion.

**`kAFE_Channel1ReadyFlag`** Channel 1 is ready to conversion.

**`kAFE_Channel2ReadyFlag`** Channel 2 is ready to conversion.

**`kAFE_Channel0ConversionCompleteFlag`** Channel 0 conversion is complete.

**`kAFE_Channel1ConversionCompleteFlag`** Channel 1 conversion is complete.

**`kAFE_Channel2ConversionCompleteFlag`** Channel 2 conversion is complete.

**`kAFE_Channel3OverflowFlag`** Channel 3 previous conversion result has not been read and new data has already arrived.

*kAFE\_Channel3ReadyFlag* Channel 3 is ready to conversion.

*kAFE\_Channel3ConversionCompleteFlag* Channel 3 conversion is complete.

## 6.6.2 anonymous enum

Enumerator

*kAFE\_Channel0InterruptEnable* Channel 0 Interrupt.

*kAFE\_Channel1InterruptEnable* Channel 1 Interrupt.

*kAFE\_Channel2InterruptEnable* Channel 2 Interrupt.

*kAFE\_Channel3InterruptEnable* Channel 3 Interrupt.

## 6.6.3 anonymous enum

Enumerator

*kAFE\_Channel0DMAEnable* Channel 0 DMA.

*kAFE\_Channel1DMAEnable* Channel 1 DMA.

*kAFE\_Channel2DMAEnable* Channel 2 DMA.

*kAFE\_Channel3DMAEnable* Channel 3 DMA.

## 6.6.4 anonymous enum

Enumerator

*kAFE\_Channel0Trigger* Channel 0 software trigger.

*kAFE\_Channel1Trigger* Channel 1 software trigger.

*kAFE\_Channel2Trigger* Channel 2 software trigger.

*kAFE\_Channel3Trigger* Channel 3 software trigger.

## 6.6.5 enum afe\_decimator\_oversample\_ratio\_t

Enumerator

*kAFE\_DecimatorOversampleRatio64* Decimator over sample ratio is 64.

*kAFE\_DecimatorOversampleRatio128* Decimator over sample ratio is 128.

*kAFE\_DecimatorOversampleRatio256* Decimator over sample ratio is 256.

*kAFE\_DecimatorOversampleRatio512* Decimator over sample ratio is 512.

*kAFE\_DecimatorOversampleRatio1024* Decimator over sample ratio is 1024.

*kAFE\_DecimatorOversampleRatio2048* Decimator over sample ratio is 2048.

## 6.6.6 enum afe\_result\_format\_t

Enumerator

- kAFE\_ResultFormatLeft* Left justified result format.
- kAFE\_ResultFormatRight* Right justified result format.

## 6.6.7 enum afe\_clock\_divider\_t

Enumerator

- kAFE\_ClockDivider1* Clock divided by 1.
- kAFE\_ClockDivider2* Clock divided by 2.
- kAFE\_ClockDivider4* Clock divided by 4.
- kAFE\_ClockDivider8* Clock divided by 8.
- kAFE\_ClockDivider16* Clock divided by 16.
- kAFE\_ClockDivider32* Clock divided by 32.
- kAFE\_ClockDivider64* Clock divided by 64.
- kAFE\_ClockDivider128* Clock divided by 128.
- kAFE\_ClockDivider256* Clock divided by 256.

## 6.6.8 enum afe\_clock\_source\_t

Enumerator

- kAFE\_ClockSource0* Modulator clock source 0.
- kAFE\_ClockSource1* Modulator clock source 1.
- kAFE\_ClockSource2* Modulator clock source 2.
- kAFE\_ClockSource3* Modulator clock source 3.

## 6.6.9 enum afe\_pga\_gain\_t

Enumerator

- kAFE\_PgaDisable* PGA disabled.
- kAFE\_PgaGain1* Input gained by 1.
- kAFE\_PgaGain2* Input gained by 2.
- kAFE\_PgaGain4* Input gained by 4.
- kAFE\_PgaGain8* Input gained by 8.
- kAFE\_PgaGain16* Input gained by 16.
- kAFE\_PgaGain32* Input gained by 32.

## 6.6.10 enum afe\_bypass\_mode\_t

Enumerator

- kAFE\_BypassInternalClockPositiveEdge*** Bypassed channel mode - internal clock selected, positive edge for registering data by the decimation filter.
- kAFE\_BypassExternalClockPositiveEdge*** Bypassed channel mode - external clock selected, positive edge for registering data by the decimation filter.
- kAFE\_BypassInternalClockNegativeEdge*** Bypassed channel mode - internal clock selected, negative edge for registering data by the decimation filter.
- kAFE\_BypassExternalClockNegativeEdge*** Bypassed channel mode - external clock selected, negative edge for registering data by the decimation filter.
- kAFE\_BypassDisable*** Normal channel mode.

## 6.7 Function Documentation

### 6.7.1 void AFE\_Init ( AFE\_Type \* *base*, const afe\_config\_t \* *config* )

This function configures the AFE module for the configuration which are shared by all channels.

Parameters

<i>base</i>	AFE peripheral base address.
<i>config</i>	Pointer to structure of "afe_config_t".

### 6.7.2 void AFE\_Deinit ( AFE\_Type \* *base* )

This function disables clock.

Parameters

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

### 6.7.3 void AFE\_GetDefaultConfig ( afe\_config\_t \* *config* )

This function fills the `afe_config_t` structure with default settings. Default value are:

```
* config->enableLowPower      = false;
* config->resultFormat       = kAFE_ResultFormatRight;
* config->clockDivider       = kAFE_ClockDivider2;
* config->clockSource        = kAFE_ClockSource1;
* config->startupCount       = 2U;
*
```

Parameters

<i>config</i>	Pointer to structure of "afe_config_t".
---------------	---

#### 6.7.4 static void AFE\_SoftwareReset ( **AFE\_Type** \* *base*, **bool** *enable* ) [**inline**], [**static**]

This function is to reset all the ADCs, PGAs, decimation filters and clock configuration bits. When asserted as "false", all ADCs, PGAs and decimation filters are disabled. Clock Configuration bits are reset. When asserted as "true", all ADCs, PGAs and decimation filters are enabled.

Parameters

<i>base</i>	AFE peripheral base address.
<i>enable</i>	Assert the reset command.

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

This function enables AFE and filter.

Parameters

<i>base</i>	AFE peripheral base address.
<i>enable</i>	Enable the AFE module or not.

#### 6.7.6 void AFE\_SetChannelConfig ( **AFE\_Type** \* *base*, **uint32\_t** *channel*, **const** **afe\_channel\_config\_t** \* *config* )

This function configures the selected AFE channel.

Parameters

<i>base</i>	AFE peripheral base address.
<i>channel</i>	AFE channel index.

<i>config</i>	Pointer to structure of "afe_channel_config_t".
---------------	---

### 6.7.7 void AFE\_GetDefaultChannelConfig ( afe\_channel\_config\_t \* *config* )

This function fills the `afe_channel_config_t` structure with default settings. Default value are:

```
* config->enableHardwareTrigger      = false;
* config->enableContinuousConversion = false;
* config->channelMode              = kAFE_Normal;
* config->decimatorOversampleRatio = kAFE_DecimatorOversampleRatio64;
* config->pgaGainSelect           = kAFE_PgaGain1;
*
```

Parameters

<i>config</i>	Pointer to structure of "afe_channel_config_t".
---------------	---

### 6.7.8 uint32\_t AFE\_GetChannelConversionValue ( AFE\_Type \* *base*, uint32\_t *channel* )

This function returns the raw conversion value of the selected channel.

Parameters

<i>base</i>	AFE peripheral base address.
<i>channel</i>	AFE channel index.

Returns

Conversion value.

Note

The returned value could be left or right adjusted according to the AFE module configuration.

### 6.7.9 static void AFE\_DoSoftwareTriggerChannel ( AFE\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function triggers the AFE conversion by executing a software command. It starts the conversion on selected channels if the software trigger option is selected for the channels.

Parameters

<i>base</i>	AFE peripheral base address.
<i>mask</i>	AFE channel mask software trigger. The parameter can be combination of the following source if defined: <ul style="list-style-type: none"> <li>• kAFE_Channel0Trigger</li> <li>• kAFE_Channel1Trigger</li> <li>• kAFE_Channel2Trigger</li> <li>• kAFE_Channel3Trigger</li> </ul>

### 6.7.10 static uint32\_t AFE\_GetChannelStatusFlags ( AFE\_Type \* *base* ) [inline], [static]

This function gets all AFE status.

Parameters

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

Returns

the mask of these status flag bits.

### 6.7.11 void AFE\_SetChannelPhaseDelayValue ( AFE\_Type \* *base*, uint32\_t *channel*, uint32\_t *value* )

This function sets the phase delays for channels. This delay is inserted before the trigger response of the decimation filters. The delay is used to provide a phase compensation between AFE channels in step of prescaled modulator clock periods.

Parameters

<i>base</i>	AFE peripheral base address.
<i>channel</i>	AFE channel index.

<i>value</i>	delay time value.
--------------	-------------------

### 6.7.12 static void AFE\_SetChannelPhasetDelayOk ( AFE\_Type \* *base* ) [inline], [static]

This function should be called after all desired channel's delay registers are loaded. Values in channel's delay registers are active after calling this function and after the conversation starts.

Parameters

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

### 6.7.13 static void AFE\_EnableChannelInterrupts ( AFE\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function enables one channel interrupt.

Parameters

<i>base</i>	AFE peripheral base address.
<i>mask</i>	AFE channel interrupt mask. The parameter can be combination of the following source if defined: <ul style="list-style-type: none"> <li>• kAFE_Channel0InterruptEnable</li> <li>• kAFE_Channel1InterruptEnable</li> <li>• kAFE_Channel2InterruptEnable</li> <li>• kAFE_Channel3InterruptEnable</li> </ul>

### 6.7.14 static void AFE\_DisableChannelInterrupts ( AFE\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function disables one channel interrupt.

Parameters

<i>base</i>	AFE peripheral base address.
<i>mask</i>	AFE channel interrupt mask. The parameter can be combination of the following source if defined: <ul style="list-style-type: none"> <li>• kAFE_Channel0InterruptEnable</li> <li>• kAFE_Channel1InterruptEnable</li> <li>• kAFE_Channel2InterruptEnable</li> <li>• kAFE_Channel3InterruptEnable</li> </ul>

### 6.7.15 static uint32\_t AFE\_GetEnabledChannelInterrupts ( AFE\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

Return the mask of these interrupt enable/disable bits.

### 6.7.16 void AFE\_EnableChannelDMA ( AFE\_Type \* *base*, uint32\_t *mask*, bool *enable* )

This function enables/disables one channel DMA request.

Parameters

<i>base</i>	AFE peripheral base address.
<i>mask</i>	AFE channel dma mask.
<i>enable</i>	Pass true to enable interrupt, false to disable. The parameter can be combination of the following source if defined: <ul style="list-style-type: none"> <li>• kAFE_Channel0DMAEnable</li> <li>• kAFE_Channel1DMAEnable</li> <li>• kAFE_Channel2DMAEnable</li> <li>• kAFE_Channel3DMAEnable</li> </ul>

# Chapter 7

## CMP: Analog Comparator Driver

### 7.1 Overview

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

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

### 7.2 Typical use case

#### 7.2.1 Polling Configuration

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

#### 7.2.2 Interrupt Configuration

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

## Data Structures

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

## Enumerations

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

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

## Driver version

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

## Initialization

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

## Advanced Features

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

## Results

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

## 7.3 Data Structure Documentation

### 7.3.1 struct cmp\_config\_t

#### Data Fields

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

#### Field Documentation

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

### 7.3.2 struct cmp\_filter\_config\_t

#### Data Fields

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

#### Field Documentation

(1) `bool cmp_filter_config_t::enableSample`

(2) `uint8_t cmp_filter_config_t::filterCount`

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

(3) `uint8_t cmp_filter_config_t::filterPeriod`

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

### 7.3.3 struct cmp\_dac\_config\_t

#### Data Fields

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

#### Field Documentation

(1) `cmp_reference_voltage_source_t cmp_dac_config_t::referenceVoltageSource`

(2) `uint8_t cmp_dac_config_t::DACValue`

Available range is 0-63.

## 7.4 Macro Definition Documentation

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

## 7.5 Enumeration Type Documentation

### 7.5.1 enum \_cmp\_interrupt\_enable

Enumerator

*kCMP\_OutputRisingInterruptEnable* Comparator interrupt enable rising.

*kCMP\_OutputFallingInterruptEnable* Comparator interrupt enable falling.

### 7.5.2 enum \_cmp\_status\_flags

Enumerator

*kCMP\_OutputRisingEventFlag* Rising-edge on the comparison output has occurred.

*kCMP\_OutputFallingEventFlag* Falling-edge on the comparison output has occurred.

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

### 7.5.3 enum cmp\_hysteresis\_mode\_t

Enumerator

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

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

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

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

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

Enumerator

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

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

## 7.6 Function Documentation

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

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

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

Parameters

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

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

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

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

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

Parameters

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

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

Parameters

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

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

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

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

Parameters

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

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

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

Parameters

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

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

Returns

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

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

## Parameters

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

# Chapter 8

## Common Driver

### 8.1 Overview

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

#### Macros

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

#### Typedefs

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

## Enumerations

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

```

kStatusGroup_BMA = 164 }

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

```

## Functions

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

## Driver version

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

## Min/max macros

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

## UINT16\_MAX/UINT32\_MAX value

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

## Suppress fallthrough warning macro

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

## 8.2 Macro Definition Documentation

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

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

### 8.2.3 #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

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

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

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

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

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

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

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

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

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

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

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

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

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

## 8.3 Typedef Documentation

### 8.3.1 typedef int32\_t status\_t

## 8.4 Enumeration Type Documentation

### 8.4.1 enum \_status\_groups

Enumerator

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

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

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

#### 8.4.2 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.

## 8.5 Function Documentation

### 8.5.1 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.
------------	-------------------

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

Parameters

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

### 8.5.3 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 9

## CRC: Cyclic Redundancy Check Driver

### 9.1 Overview

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

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

### 9.2 CRC Driver Initialization and Configuration

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

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

### 9.3 CRC Write Data

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

### 9.4 CRC Get Checksum

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

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

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

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

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

## 9.5 Comments about API usage in RTOS

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

The triplets

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

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

## Data Structures

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

## Macros

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

## Enumerations

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

## Functions

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

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

## Driver version

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

## 9.6 Data Structure Documentation

### 9.6.1 struct crc\_config\_t

This structure holds the configuration for the CRC protocol.

#### Data Fields

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

#### Field Documentation

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

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

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

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

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

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

## 9.7 Macro Definition Documentation

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

Version 2.0.3.

Current version: 2.0.3

Change log:

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

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

Use CRC16-CCIT-FALSE as default.

## 9.8 Enumeration Type Documentation

### 9.8.1 enum crc\_bits\_t

Enumerator

*kCrcBits16* Generate 16-bit CRC code.

*kCrcBits32* Generate 32-bit CRC code.

### 9.8.2 enum crc\_result\_t

Enumerator

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

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

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

## 9.9 Function Documentation

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

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

Parameters

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

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

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

Parameters

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

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

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

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

Parameters

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

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

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

Parameters

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

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

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

Parameters

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

Returns

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

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

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

Parameters

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

Returns

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

# Chapter 10

## DMA: Direct Memory Access Controller Driver

### 10.1 Overview

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

### 10.2 Typical use case

#### 10.2.1 DMA Operation

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

### Data Structures

- struct `dma_transfer_config_t`  
*DMA transfer configuration structure. [More...](#)*
- struct `dma_channel_link_config_t`  
*DMA transfer configuration structure. [More...](#)*
- struct `dma_handle_t`  
*DMA DMA handle structure. [More...](#)*

### Typedefs

- `typedef void(* dma_callback )(struct _dma_handle *handle, void *userData)`  
*Callback function prototype for the DMA driver.*

### Enumerations

- enum {  
  `kDMA_TransactionsBCRFlag` = DMA\_DSR\_BCR\_BCR\_MASK,  
  `kDMA_TransactionsDoneFlag` = DMA\_DSR\_BCR\_DONE\_MASK,  
  `kDMA_TransactionsBusyFlag` = DMA\_DSR\_BCR\_BSY\_MASK,  
  `kDMA_TransactionsRequestFlag` = DMA\_DSR\_BCR\_REQ\_MASK,  
  `kDMA_BusErrorOnDestinationFlag` = DMA\_DSR\_BCR\_BED\_MASK,  
  `kDMA_BusErrorOnSourceFlag` = DMA\_DSR\_BCR\_BES\_MASK,  
  `kDMA_ConfigurationErrorFlag` = DMA\_DSR\_BCR\_CE\_MASK }  
    *\_dma\_channel\_status\_flags status flag for the DMA driver.*
- enum `dma_transfer_size_t` {  
  `kDMA_Transfersize32bits` = 0x0U,  
  `kDMA_Transfersize8bits`,  
  `kDMA_Transfersize16bits` }  
    *DMA transfer size type.*

- enum `dma_modulo_t` {
   
    `kDMA_ModuloDisable` = 0x0U,
   
    `kDMA_Modulo16Bytes`,
   
    `kDMA_Modulo32Bytes`,
   
    `kDMA_Modulo64Bytes`,
   
    `kDMA_Modulo128Bytes`,
   
    `kDMA_Modulo256Bytes`,
   
    `kDMA_Modulo512Bytes`,
   
    `kDMA_Modulo1KBytes`,
   
    `kDMA_Modulo2KBytes`,
   
    `kDMA_Modulo4KBytes`,
   
    `kDMA_Modulo8KBytes`,
   
    `kDMA_Modulo16KBytes`,
   
    `kDMA_Modulo32KBytes`,
   
    `kDMA_Modulo64KBytes`,
   
    `kDMA_Modulo128KBytes`,
   
    `kDMA_Modulo256KBytes` }

*Configuration type for the DMA modulo.*

- enum `dma_channel_link_type_t` {
   
    `kDMA_ChannelLinkDisable` = 0x0U,
   
    `kDMA_ChannelLinkChannel1AndChannel2`,
   
    `kDMA_ChannelLinkChannel1`,
   
    `kDMA_ChannelLinkChannel1AfterBCR0` }

*DMA channel link type.*

- enum `dma_transfer_type_t` {
   
    `kDMA_MemoryToMemory` = 0x0U,
   
    `kDMA_PeripheralToMemory`,
   
    `kDMA_MemoryToPeripheral` }

*DMA transfer type.*

- enum `dma_transfer_options_t` {
   
    `kDMA_NoOptions` = 0x0U,
   
    `kDMA_EnableInterrupt` }

*DMA transfer options.*

- enum `dma_addr_increment_t` {
   
    `kDMA_AddrNoIncrement` = 0x0U,
   
    `kDMA_AddrIncrementPerTransferWidth` = 0x1U }
- dma addre increment type*
- enum { `kStatus_DMA_Busy` = MAKE\_STATUS(kStatusGroup\_DMA, 0) }
- \_dma\_transfer\_status DMA transfer status*

## Driver version

- #define `FSL_DMA_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 1))

*DMA driver version 2.1.1.*

## DMA Initialization and De-initialization

- void **DMA\_Init** (DMA\_Type \*base)  
*Initializes the DMA peripheral.*
- void **DMA\_Deinit** (DMA\_Type \*base)  
*Deinitializes the DMA peripheral.*

## DMA Channel Operation

- void **DMA\_ResetChannel** (DMA\_Type \*base, uint32\_t channel)  
*Resets the DMA channel.*
- void **DMA\_SetTransferConfig** (DMA\_Type \*base, uint32\_t channel, const **dma\_transfer\_config\_t** \*config)  
*Configures the DMA transfer attribute.*
- void **DMA\_SetChannelLinkConfig** (DMA\_Type \*base, uint32\_t channel, const **dma\_channel\_link\_config\_t** \*config)  
*Configures the DMA channel link feature.*
- static void **DMA\_SetSourceAddress** (DMA\_Type \*base, uint32\_t channel, uint32\_t srcAddr)  
*Sets the DMA source address for the DMA transfer.*
- static void **DMA\_SetDestinationAddress** (DMA\_Type \*base, uint32\_t channel, uint32\_t destAddr)  
*Sets the DMA destination address for the DMA transfer.*
- static void **DMA\_SetTransferSize** (DMA\_Type \*base, uint32\_t channel, uint32\_t size)  
*Sets the DMA transfer size for the DMA transfer.*
- void **DMA\_SetModulo** (DMA\_Type \*base, uint32\_t channel, **dma\_modulo\_t** srcModulo, **dma\_modulo\_t** destModulo)  
*Sets the DMA modulo for the DMA transfer.*
- static void **DMA\_EnableCycleSteal** (DMA\_Type \*base, uint32\_t channel, bool enable)  
*Enables the DMA cycle steal for the DMA transfer.*
- static void **DMA\_EnableAutoAlign** (DMA\_Type \*base, uint32\_t channel, bool enable)  
*Enables the DMA auto align for the DMA transfer.*
- static void **DMA\_EnableAsyncRequest** (DMA\_Type \*base, uint32\_t channel, bool enable)  
*Enables the DMA async request for the DMA transfer.*
- static void **DMA\_EnableInterrupts** (DMA\_Type \*base, uint32\_t channel)  
*Enables an interrupt for the DMA transfer.*
- static void **DMA\_DisableInterrupts** (DMA\_Type \*base, uint32\_t channel)  
*Disables an interrupt for the DMA transfer.*

## DMA Channel Transfer Operation

- static void **DMA\_EnableChannelRequest** (DMA\_Type \*base, uint32\_t channel)  
*Enables the DMA hardware channel request.*
- static void **DMA\_DisableChannelRequest** (DMA\_Type \*base, uint32\_t channel)  
*Disables the DMA hardware channel request.*
- static void **DMA\_TriggerChannelStart** (DMA\_Type \*base, uint32\_t channel)  
*Starts the DMA transfer with a software trigger.*
- static void **DMA\_EnableAutoStopRequest** (DMA\_Type \*base, uint32\_t channel, bool enable)  
*Starts the DMA enable/disable auto disable request.*

## DMA Channel Status Operation

- static uint32\_t **DMA\_GetRemainingBytes** (DMA\_Type \*base, uint32\_t channel)

- Gets the remaining bytes of the current DMA transfer.  
static uint32\_t **DMA\_GetChannelStatusFlags** (DMA\_Type \*base, uint32\_t channel)  
Gets the DMA channel status flags.
- static void **DMA\_ClearChannelStatusFlags** (DMA\_Type \*base, uint32\_t channel, uint32\_t mask)  
Clears the DMA channel status flags.

## DMA Channel Transactional Operation

- void **DMA\_CreateHandle** (dma\_handle\_t \*handle, DMA\_Type \*base, uint32\_t channel)  
Creates the DMA handle.
- void **DMA\_SetCallback** (dma\_handle\_t \*handle, dma\_callback callback, void \*userData)  
Sets the DMA callback function.
- void **DMA\_PreparesTransferConfig** (dma\_transfer\_config\_t \*config, void \*srcAddr, uint32\_t srcWidth, void \*destAddr, uint32\_t destWidth, uint32\_t transferBytes, dma\_addr\_increment\_t srcIncrement, dma\_addr\_increment\_t destIncrement)  
Prepares the DMA transfer configuration structure.
- void **DMA\_PreparesTransfer** (dma\_transfer\_config\_t \*config, void \*srcAddr, uint32\_t srcWidth, void \*destAddr, uint32\_t destWidth, uint32\_t transferBytes, dma\_transfer\_type\_t type)  
Prepares the DMA transfer configuration structure.
- status\_t **DMA\_SubmitTransfer** (dma\_handle\_t \*handle, const dma\_transfer\_config\_t \*config, uint32\_t options)  
Submits the DMA transfer request.
- static void **DMA\_StartTransfer** (dma\_handle\_t \*handle)  
DMA starts a transfer.
- static void **DMA\_StopTransfer** (dma\_handle\_t \*handle)  
DMA stops a transfer.
- void **DMA\_AbortTransfer** (dma\_handle\_t \*handle)  
DMA aborts a transfer.
- void **DMA\_HandleIRQ** (dma\_handle\_t \*handle)  
DMA IRQ handler for current transfer complete.

## 10.3 Data Structure Documentation

### 10.3.1 struct dma\_transfer\_config\_t

#### Data Fields

- uint32\_t **srcAddr**  
DMA transfer source address.
- uint32\_t **destAddr**  
DMA destination address.
- bool **enableSrcIncrement**  
Source address increase after each transfer.
- **dma\_transfer\_size\_t srcSize**  
Source transfer size unit.
- bool **enableDestIncrement**  
Destination address increase after each transfer.
- **dma\_transfer\_size\_t destSize**  
Destination transfer unit.
- uint32\_t **transferSize**

*The number of bytes to be transferred.*

#### Field Documentation

- (1) `uint32_t dma_transfer_config_t::srcAddr`
- (2) `uint32_t dma_transfer_config_t::destAddr`
- (3) `bool dma_transfer_config_t::enableSrcIncrement`
- (4) `dma_transfer_size_t dma_transfer_config_t::srcSize`
- (5) `bool dma_transfer_config_t::enableDestIncrement`
- (6) `dma_transfer_size_t dma_transfer_config_t::destSize`
- (7) `uint32_t dma_transfer_config_t::transferSize`

### 10.3.2 struct `dma_channel_link_config_t`

#### Data Fields

- `dma_channel_link_type_t linkType`  
*Channel link type.*
- `uint32_t channel1`  
*The index of channel 1.*
- `uint32_t channel2`  
*The index of channel 2.*

#### Field Documentation

- (1) `dma_channel_link_type_t dma_channel_link_config_t::linkType`
- (2) `uint32_t dma_channel_link_config_t::channel1`
- (3) `uint32_t dma_channel_link_config_t::channel2`

### 10.3.3 struct `dma_handle_t`

#### Data Fields

- `DMA_Type * base`  
*DMA peripheral address.*
- `uint8_t channel`  
*DMA channel used.*
- `dma_callback callback`  
*DMA callback function.*
- `void * userData`  
*Callback parameter.*

**Field Documentation**

- (1) DMA\_Type\* dma\_handle\_t::base
- (2) uint8\_t dma\_handle\_t::channel
- (3) dma\_callback dma\_handle\_t::callback
- (4) void\* dma\_handle\_t::userData

**10.4 Macro Definition Documentation**

**10.4.1 #define FSL\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))**

**10.5 Typedef Documentation**

**10.5.1 typedef void(\* dma\_callback)(struct \_dma\_handle \*handle, void \*userData)**

**10.6 Enumeration Type Documentation****10.6.1 anonymous enum**

Enumerator

*kDMA\_TransactionsBCRFlag* Contains the number of bytes yet to be transferred for a given block.

*kDMA\_TransactionsDoneFlag* Transactions Done.

*kDMA\_TransactionsBusyFlag* Transactions Busy.

*kDMA\_TransactionsRequestFlag* Transactions Request.

*kDMA\_BusErrorOnDestinationFlag* Bus Error on Destination.

*kDMA\_BusErrorOnSourceFlag* Bus Error on Source.

*kDMA\_ConfigurationErrorFlag* Configuration Error.

**10.6.2 enum dma\_transfer\_size\_t**

Enumerator

*kDMA\_TransferSize32bits* 32 bits are transferred for every read/write

*kDMA\_TransferSize8bits* 8 bits are transferred for every read/write

*kDMA\_TransferSize16bits* 16b its are transferred for every read/write

**10.6.3 enum dma\_modulo\_t**

Enumerator

*kDMA\_ModuloDisable* Buffer disabled.

<i>kDMA_Modulo16Bytes</i>	Circular buffer size is 16 bytes.
<i>kDMA_Modulo32Bytes</i>	Circular buffer size is 32 bytes.
<i>kDMA_Modulo64Bytes</i>	Circular buffer size is 64 bytes.
<i>kDMA_Modulo128Bytes</i>	Circular buffer size is 128 bytes.
<i>kDMA_Modulo256Bytes</i>	Circular buffer size is 256 bytes.
<i>kDMA_Modulo512Bytes</i>	Circular buffer size is 512 bytes.
<i>kDMA_Modulo1KBytes</i>	Circular buffer size is 1 KB.
<i>kDMA_Modulo2KBytes</i>	Circular buffer size is 2 KB.
<i>kDMA_Modulo4KBytes</i>	Circular buffer size is 4 KB.
<i>kDMA_Modulo8KBytes</i>	Circular buffer size is 8 KB.
<i>kDMA_Modulo16KBytes</i>	Circular buffer size is 16 KB.
<i>kDMA_Modulo32KBytes</i>	Circular buffer size is 32 KB.
<i>kDMA_Modulo64KBytes</i>	Circular buffer size is 64 KB.
<i>kDMA_Modulo128KBytes</i>	Circular buffer size is 128 KB.
<i>kDMA_Modulo256KBytes</i>	Circular buffer size is 256 KB.

#### 10.6.4 enum dma\_channel\_link\_type\_t

Enumerator

<i>kDMA_ChannelLinkDisable</i>	No channel link.
<i>kDMA_ChannelLinkChannel1AndChannel2</i>	Perform a link to channel LCH1 after each cycle-steal transfer, followed by a link to LCH2 after the BCR decrements to 0.
<i>kDMA_ChannelLinkChannel1</i>	Perform a link to LCH1 after each cycle-steal transfer.
<i>kDMA_ChannelLinkChannel1AfterBCR0</i>	Perform a link to LCH1 after the BCR decrements.

#### 10.6.5 enum dma\_transfer\_type\_t

Enumerator

<i>kDMA_MemoryToMemory</i>	Memory to Memory transfer.
<i>kDMA_PeripheralToMemory</i>	Peripheral to Memory transfer.
<i>kDMA_MemoryToPeripheral</i>	Memory to Peripheral transfer.

#### 10.6.6 enum dma\_transfer\_options\_t

Enumerator

<i>kDMA_NoOptions</i>	Transfer without options.
<i>kDMA_EnableInterrupt</i>	Enable interrupt while transfer complete.

### 10.6.7 enum dma\_addr\_increment\_t

Enumerator

***kDMA\_AddrNoIncrement*** Transfer address not increment.

***kDMA\_AddrIncrementPerTransferWidth*** Transfer address increment per transfer width.

### 10.6.8 anonymous enum

Enumerator

***kStatus\_DMA\_Busy*** DMA is busy.

## 10.7 Function Documentation

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

This function ungates the DMA clock.

Parameters

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

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

This function gates the DMA clock.

Parameters

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

### 10.7.3 void DMA\_ResetChannel ( DMA\_Type \* *base*, uint32\_t *channel* )

Sets all register values to reset values and enables the cycle steal and auto stop channel request features.

Parameters

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

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

#### 10.7.4 void DMA\_SetTransferConfig ( DMA\_Type \* *base*, uint32\_t *channel*, const dma\_transfer\_config\_t \* *config* )

This function configures the transfer attribute including the source address, destination address, transfer size, and so on. This example shows how to set up the `dma_transfer_config_t` parameters and how to call the DMA\_ConfigBasicTransfer function.

```
*     dma_transfer_config_t transferConfig;
*     memset(&transferConfig, 0, sizeof(transferConfig));
*     transferConfig.srcAddr = (uint32_t)srcAddr;
*     transferConfig.destAddr = (uint32_t)destAddr;
*     transferConfig.enableSrcIncrement = true;
*     transferConfig.enableDestIncrement = true;
*     transferConfig.srcSize = kDMA_Transfersize32bits;
*     transferConfig.destSize = kDMA_Transfersize32bits;
*     transferConfig.transferSize = sizeof(uint32_t) * BUFF_LENGTH;
*     DMA_SetTransferConfig(DMA0, 0, &transferConfig);
*
```

##### Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.
<i>config</i>	Pointer to the DMA transfer configuration structure.

#### 10.7.5 void DMA\_SetChannelLinkConfig ( DMA\_Type \* *base*, uint32\_t *channel*, const dma\_channel\_link\_config\_t \* *config* )

This function allows DMA channels to have their transfers linked. The current DMA channel triggers a DMA request to the linked channels (LCH1 or LCH2) depending on the channel link type. Perform a link to channel LCH1 after each cycle-steal transfer followed by a link to LCH2 after the BCR decrements to 0 if the type is `kDMA_ChannelLinkChannel1AndChannel2`. Perform a link to LCH1 after each cycle-steal transfer if the type is `kDMA_ChannelLinkChannel1`. Perform a link to LCH1 after the BCR decrements to 0 if the type is `kDMA_ChannelLinkChannel1AfterBCR0`.

##### Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.
<i>config</i>	Pointer to the channel link configuration structure.

#### 10.7.6 static void DMA\_SetSourceAddress ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *srcAddr* ) [inline], [static]

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.
<i>srcAddr</i>	DMA source address.

#### 10.7.7 static void DMA\_SetDestinationAddress ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *destAddr* ) [inline], [static]

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.
<i>destAddr</i>	DMA destination address.

#### 10.7.8 static void DMA\_SetTransferSize ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *size* ) [inline], [static]

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.
<i>size</i>	The number of bytes to be transferred.

### 10.7.9 void DMA\_SetModulo ( DMA\_Type \* *base*, uint32\_t *channel*, dma\_modulo\_t *srcModulo*, dma\_modulo\_t *destModulo* )

This function defines a specific address range specified to be the value after (SAR + SSIZE)/(DAR + DSIZE) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.
<i>srcModulo</i>	source address modulo.
<i>destModulo</i>	destination address modulo.

### 10.7.10 static void DMA\_EnableCycleSteal ( DMA\_Type \* *base*, uint32\_t *channel*, bool *enable* ) [inline], [static]

If the cycle steal feature is enabled (true), the DMA controller forces a single read/write transfer per request, or it continuously makes read/write transfers until the BCR decrements to 0.

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.
<i>enable</i>	The command for enable (true) or disable (false).

### 10.7.11 static void DMA\_EnableAutoAlign ( DMA\_Type \* *base*, uint32\_t *channel*, bool *enable* ) [inline], [static]

If the auto align feature is enabled (true), the appropriate address register increments regardless of DINC or SINC.

Parameters

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

<i>channel</i>	DMA channel number.
<i>enable</i>	The command for enable (true) or disable (false).

#### 10.7.12 static void DMA\_EnableAsyncRequest ( DMA\_Type \* *base*, uint32\_t *channel*, bool *enable* ) [inline], [static]

If the async request feature is enabled (true), the DMA supports asynchronous DREQs while the MCU is in stop mode.

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.
<i>enable</i>	The command for enable (true) or disable (false).

#### 10.7.13 static void DMA\_EnableInterrupts ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters

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

#### 10.7.14 static void DMA\_DisableInterrupts ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters

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

#### 10.7.15 static void DMA\_EnableChannelRequest ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	The DMA channel number.

#### 10.7.16 static void DMA\_DisableChannelRequest ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters

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

#### 10.7.17 static void DMA\_TriggerChannelStart ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This function starts only one read/write iteration.

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	The DMA channel number.

#### 10.7.18 static void DMA\_EnableAutoStopRequest ( DMA\_Type \* *base*, uint32\_t *channel*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	The DMA channel number.
<i>enable</i>	true is enable, false is disable.

#### 10.7.19 static uint32\_t DMA\_GetRemainingBytes ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters

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

Returns

The number of bytes which have not been transferred yet.

#### 10.7.20 static uint32\_t DMA\_GetChannelStatusFlags ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters

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

Returns

The mask of the channel status. Use the \_dma\_channel\_status\_flags type to decode the return 32 bit variables.

#### 10.7.21 static void DMA\_ClearChannelStatusFlags ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.
<i>mask</i>	The mask of the channel status to be cleared. Use the defined _dma_channel_status_flags type.

#### 10.7.22 void DMA\_CreateHandle ( dma\_handle\_t \* *handle*, DMA\_Type \* *base*, uint32\_t *channel* )

This function is called first if using the transactional API for the DMA. This function initializes the internal state of the DMA handle.

Parameters

<i>handle</i>	DMA handle pointer. The DMA handle stores callback function and parameters.
<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.

#### **10.7.23 void DMA\_SetCallback ( *dma\_handle\_t \* handle*, *dma\_callback callback*, *void \* userData* )**

This callback is called in the DMA IRQ handler. Use the callback to do something after the current transfer complete.

Parameters

<i>handle</i>	DMA handle pointer.
<i>callback</i>	DMA callback function pointer.
<i>userData</i>	Parameter for callback function. If it is not needed, just set to NULL.

#### **10.7.24 void DMA\_PrepTransferConfig ( *dma\_transfer\_config\_t \* config*, *void \* srcAddr*, *uint32\_t srcWidth*, *void \* destAddr*, *uint32\_t destWidth*, *uint32\_t transferBytes*, *dma\_addr\_increment\_t srcIncrement*, *dma\_addr\_increment\_t destIncrement* )**

This function prepares the transfer configuration structure according to the user input. The difference between this function and DMA\_PrepTransfer is that this function expose the address increment parameter to application, but in DMA\_PrepTransfer, only parts of the address increment option can be selected by *dma\_transfer\_type\_t*.

Parameters

<i>config</i>	Pointer to the user configuration structure of type <a href="#">dma_transfer_config_t</a> .
<i>srcAddr</i>	DMA transfer source address.
<i>srcWidth</i>	DMA transfer source address width (byte).
<i>destAddr</i>	DMA transfer destination address.
<i>destWidth</i>	DMA transfer destination address width (byte).
<i>transferBytes</i>	DMA transfer bytes to be transferred.
<i>srcIncrement</i>	source address increment type.
<i>destIncrement</i>	dest address increment type.

**10.7.25 void DMA\_PrepTransfer ( *dma\_transfer\_config\_t* \* *config*, *void* \* *srcAddr*, *uint32\_t* *srcWidth*, *void* \* *destAddr*, *uint32\_t* *destWidth*, *uint32\_t* *transferBytes*, *dma\_transfer\_type\_t* *type* )**

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

Parameters

<i>config</i>	Pointer to the user configuration structure of type <a href="#">dma_transfer_config_t</a> .
<i>srcAddr</i>	DMA transfer source address.
<i>srcWidth</i>	DMA transfer source address width (byte).
<i>destAddr</i>	DMA transfer destination address.
<i>destWidth</i>	DMA transfer destination address width (byte).
<i>transferBytes</i>	DMA transfer bytes to be transferred.
<i>type</i>	DMA transfer type.

**10.7.26 status\_t DMA\_SubmitTransfer ( *dma\_handle\_t* \* *handle*, *const dma\_transfer\_config\_t* \* *config*, *uint32\_t* *options* )**

This function submits the DMA transfer request according to the transfer configuration structure.

Parameters

<i>handle</i>	DMA handle pointer.
<i>config</i>	Pointer to DMA transfer configuration structure.
<i>options</i>	Additional configurations for transfer. Use the defined <i>dma_transfer_options_t</i> type.

Return values

<i>kStatus_DMA_Success</i>	It indicates that the DMA submit transfer request succeeded.
<i>kStatus_DMA_Busy</i>	It indicates that the DMA is busy. Submit transfer request is not allowed.

Note

This function can't process multi transfer request.

**10.7.27 static void DMA\_StartTransfer ( `dma_handle_t * handle` ) [inline],  
[static]**

This function enables the channel request. Call this function after submitting a transfer request.

Parameters

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

Return values

<i>kStatus_DMA_Success</i>	It indicates that the DMA start transfer succeed.
<i>kStatus_DMA_Busy</i>	It indicates that the DMA has started a transfer.

#### 10.7.28 static void DMA\_StopTransfer ( **dma\_handle\_t \* handle** ) [inline], [static]

This function disables the channel request to stop a DMA transfer. The transfer can be resumed by calling the DMA\_StartTransfer.

Parameters

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

#### 10.7.29 void DMA\_AbortTransfer ( **dma\_handle\_t \* handle** )

This function disables the channel request and clears all status bits. Submit another transfer after calling this API.

Parameters

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

#### 10.7.30 void DMA\_HandleIRQ ( **dma\_handle\_t \* handle** )

This function clears the channel interrupt flag and calls the callback function if it is not NULL.

Parameters

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

# Chapter 11

## DMAMUX: Direct Memory Access Multiplexer Driver

### 11.1 Overview

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

### 11.2 Typical use case

#### 11.2.1 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.*

### 11.3 Macro Definition Documentation

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

### 11.4 Function Documentation

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

This function ungates the DMAMUX clock.

Parameters

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

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

This function gates the DMAMUX clock.

Parameters

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

#### 11.4.3 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.

#### 11.4.4 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.
----------------	------------------------

#### 11.4.5 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.

#### 11.4.6 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.

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

This function disables the DMAMUX period trigger.

Parameters

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

# Chapter 12

## EWM: External Watchdog Monitor Driver

### 12.1 Overview

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

### 12.2 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.*

## 12.3 Data Structure Documentation

### 12.3.1 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.*

## 12.4 Macro Definition Documentation

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

## 12.5 Enumeration Type Documentation

### 12.5.1 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].

## 12.5.2 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.

## 12.5.3 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.

## 12.6 Function Documentation

### 12.6.1 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

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

This function is used to shut down the EWM.

Parameters

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

### 12.6.3 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](#)

### 12.6.4 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>

### 12.6.5 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>

## 12.6.6 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.

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

This function resets the EWM counter to zero.

## Parameters

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

# Chapter 13

## C90TFS Flash Driver

### 13.1 Overview

The flash provides the C90TFS Flash driver of Kinetis devices with the C90TFS Flash module inside. The flash driver provides general APIs to handle specific operations on C90TFS/FTFx Flash module. The user can use those APIs directly in the application. In addition, it provides internal functions called by the driver. Although these functions are not meant to be called from the user's application directly, the APIs can still be used.

### Modules

- [Ftftx CACHE Driver](#)
- [Ftftx FLASH Driver](#)
- [Ftftx FLEXNVM Driver](#)
- [ftfx controller](#)
- [ftfx feature](#)

## 13.2 Ftftx FLASH Driver

### 13.2.1 Overview

#### Data Structures

- union `pflash_prot_status_t`  
*PFlash protection status.* [More...](#)
- struct `flash_config_t`  
*Flash driver state information.* [More...](#)

#### Enumerations

- enum `flash_prot_state_t` {
   
`kFLASH_ProtectionStateUnprotected`,
   
`kFLASH_ProtectionStateProtected`,
   
`kFLASH_ProtectionStateMixed` }
   
*Enumeration for the three possible flash protection levels.*
- enum `flash_property_tag_t` {
   
`kFLASH_PropertyPflash0SectorSize` = 0x00U,
   
`kFLASH_PropertyPflash0TotalSize` = 0x01U,
   
`kFLASH_PropertyPflash0BlockSize` = 0x02U,
   
`kFLASH_PropertyPflash0BlockCount` = 0x03U,
   
`kFLASH_PropertyPflash0BlockBaseAddr` = 0x04U,
   
`kFLASH_PropertyPflash0FacSupport` = 0x05U,
   
`kFLASH_PropertyPflash0AccessSegmentSize` = 0x06U,
   
`kFLASH_PropertyPflash0AccessSegmentCount` = 0x07U,
   
`kFLASH_PropertyPflash1SectorSize` = 0x10U,
   
`kFLASH_PropertyPflash1TotalSize` = 0x11U,
   
`kFLASH_PropertyPflash1BlockSize` = 0x12U,
   
`kFLASH_PropertyPflash1BlockCount` = 0x13U,
   
`kFLASH_PropertyPflash1BlockBaseAddr` = 0x14U,
   
`kFLASH_PropertyPflash1FacSupport` = 0x15U,
   
`kFLASH_PropertyPflash1AccessSegmentSize` = 0x16U,
   
`kFLASH_PropertyPflash1AccessSegmentCount` = 0x17U,
   
`kFLASH_PropertyFlexRamBlockBaseAddr` = 0x20U,
   
`kFLASH_PropertyFlexRamTotalSize` = 0x21U }
   
*Enumeration for various flash properties.*

#### Flash version

- #define `FSL_FLASH_DRIVER_VERSION` (`MAKE_VERSION(3U, 1U, 2U)`)
   
*Flash driver version for SDK.*
- #define `FSL_FLASH_DRIVER_VERSION_ROM` (`MAKE_VERSION(3U, 0U, 0U)`)
   
*Flash driver version for ROM.*

## Initialization

- **status\_t FLASH\_Init (flash\_config\_t \*config)**  
*Initializes the global flash properties structure members.*

## Erasing

- **status\_t FLASH\_Erase (flash\_config\_t \*config, uint32\_t start, uint32\_t lengthInBytes, uint32\_t key)**  
*Erases the Dflash sectors encompassed by parameters passed into function.*
- **status\_t FLASH\_EraseSectorNonBlocking (flash\_config\_t \*config, uint32\_t start, uint32\_t key)**  
*Erases the Dflash sectors encompassed by parameters passed into function.*
- **status\_t FLASH\_EraseAll (flash\_config\_t \*config, uint32\_t key)**  
*Erases entire flexnvm.*

## Programming

- **status\_t FLASH\_Program (flash\_config\_t \*config, uint32\_t start, uint8\_t \*src, uint32\_t lengthInBytes)**  
*Programs flash with data at locations passed in through parameters.*
- **status\_t FLASH\_ProgramOnce (flash\_config\_t \*config, uint32\_t index, uint8\_t \*src, uint32\_t lengthInBytes)**  
*Program the Program-Once-Field through parameters.*

## Reading

- **status\_t FLASH\_ReadResource (flash\_config\_t \*config, uint32\_t start, uint8\_t \*dst, uint32\_t lengthInBytes, ftx\_read\_resource\_opt\_t option)**  
*Reads the resource with data at locations passed in through parameters.*
- **status\_t FLASH\_ReadOnce (flash\_config\_t \*config, uint32\_t index, uint8\_t \*dst, uint32\_t lengthInBytes)**  
*Reads the Program Once Field through parameters.*

## Verification

- **status\_t FLASH\_VerifyErase (flash\_config\_t \*config, uint32\_t start, uint32\_t lengthInBytes, ftx\_margin\_value\_t margin)**  
*Verifies an erasure of the desired flash area at a specified margin level.*
- **status\_t FLASH\_VerifyEraseAll (flash\_config\_t \*config, ftx\_margin\_value\_t margin)**  
*Verifies erasure of the entire flash at a specified margin level.*
- **status\_t FLASH\_VerifyProgram (flash\_config\_t \*config, uint32\_t start, uint32\_t lengthInBytes, const uint8\_t \*expectedData, ftx\_margin\_value\_t margin, uint32\_t \*failedAddress, uint32\_t \*failedData)**  
*Verifies programming of the desired flash area at a specified margin level.*

## Security

- `status_t FLASH_GetSecurityState (flash_config_t *config, ftfx_security_state_t *state)`  
*Returns the security state via the pointer passed into the function.*
- `status_t FLASH_SecurityBypass (flash_config_t *config, const uint8_t *backdoorKey)`  
*Allows users to bypass security with a backdoor key.*

## Protection

- `status_t FLASH_IsProtected (flash_config_t *config, uint32_t start, uint32_t lengthInBytes, flash_prot_state_t *protection_state)`  
*Returns the protection state of the desired flash area via the pointer passed into the function.*
- `status_t FLASH_PflashSetProtection (flash_config_t *config, pflash_prot_status_t *protectStatus)`  
*Sets the PFlash Protection to the intended protection status.*
- `status_t FLASH_PflashGetProtection (flash_config_t *config, pflash_prot_status_t *protectStatus)`  
*Gets the PFlash protection status.*

## Properties

- `status_t FLASHGetProperty (flash_config_t *config, flash_property_tag_t whichProperty, uint32_t *value)`  
*Returns the desired flash property.*

## commandStatus

- `status_t FLASH_GetCommandState (void)`  
*Get previous command status.*

### 13.2.2 Data Structure Documentation

#### 13.2.2.1 union pflash\_prot\_status\_t

##### Data Fields

- `uint32_t protl`  
*PROT[31:0].*
- `uint32_t proth`  
*PROT[63:32].*
- `uint8_t protsl`  
*PROTS[7:0].*
- `uint8_t protsh`  
*PROTS[15:8].*

## Field Documentation

- (1) `uint32_t pflash_prot_status_t::protl`
- (2) `uint32_t pflash_prot_status_t::proth`
- (3) `uint8_t pflash_prot_status_t::protsl`
- (4) `uint8_t pflash_prot_status_t::protsh`

### 13.2.2.2 struct flash\_config\_t

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

## 13.2.3 Macro Definition Documentation

### 13.2.3.1 #define FSL\_FLASH\_DRIVER\_VERSION (MAKE\_VERSION(3U, 1U, 2U))

Version 3.1.2.

### 13.2.3.2 #define FSL\_FLASH\_DRIVER\_VERSION\_ROM (MAKE\_VERSION(3U, 0U, 0U))

Version 3.0.0.

## 13.2.4 Enumeration Type Documentation

### 13.2.4.1 enum flash\_prot\_state\_t

Enumerator

*kFLASH\_ProtectionStateUnprotected* Flash region is not protected.

*kFLASH\_ProtectionStateProtected* Flash region is protected.

*kFLASH\_ProtectionStateMixed* Flash is mixed with protected and unprotected region.

### 13.2.4.2 enum flash\_property\_tag\_t

Enumerator

*kFLASH\_PropertyPflash0SectorSize* Pflash sector size property.

*kFLASH\_PropertyPflash0TotalSize* Pflash total size property.

*kFLASH\_PropertyPflash0BlockSize* Pflash block size property.

*kFLASH\_PropertyPflash0BlockCount* Pflash block count property.

*kFLASH\_PropertyPflash0BlockBaseAddr* Pflash block base address property.  
*kFLASH\_PropertyPflash0FacSupport* Pflash fac support property.  
*kFLASH\_PropertyPflash0AccessSegmentSize* Pflash access segment size property.  
*kFLASH\_PropertyPflash0AccessSegmentCount* Pflash access segment count property.  
*kFLASH\_PropertyPflash1SectorSize* Pflash sector size property.  
*kFLASH\_PropertyPflash1TotalSize* Pflash total size property.  
*kFLASH\_PropertyPflash1BlockSize* Pflash block size property.  
*kFLASH\_PropertyPflash1BlockCount* Pflash block count property.  
*kFLASH\_PropertyPflash1BlockBaseAddr* Pflash block base address property.  
*kFLASH\_PropertyPflash1FacSupport* Pflash fac support property.  
*kFLASH\_PropertyPflash1AccessSegmentSize* Pflash access segment size property.  
*kFLASH\_PropertyPflash1AccessSegmentCount* Pflash access segment count property.  
*kFLASH\_PropertyFlexRamBlockBaseAddr* FlexRam block base address property.  
*kFLASH\_PropertyFlexRamTotalSize* FlexRam total size property.

### 13.2.5 Function Documentation

#### 13.2.5.1 status\_t **FLASH\_Init** ( **flash\_config\_t \* config** )

This function checks and initializes the Flash module for the other Flash APIs.

Parameters

<i>config</i>	Pointer to the storage for the driver runtime state.
---------------	--

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Partition-StatusUpdateFailure</i>	Failed to update the partition status.

#### 13.2.5.2 status\_t **FLASH\_Erase** ( **flash\_config\_t \* config, uint32\_t start, uint32\_t lengthInBytes, uint32\_t key** )

This function erases the appropriate number of flash sectors based on the desired start address and length.

## Parameters

<i>config</i>	The pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words) to be erased. Must be word-aligned.
<i>key</i>	The value used to validate all flash erase APIs.

## Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the appropriate number of flash sectors based on the desired start address and length were erased successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	The parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_AddressError</i>	The address is out of range.
<i>kStatus_FTFx_EraseKeyError</i>	The API erase key is invalid.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

**13.2.5.3 status\_t FLASH\_EraseSectorNonBlocking ( *flash\_config\_t \* config, uint32\_t start, uint32\_t key* )**

This function erases one flash sector size based on the start address, and it is executed asynchronously.

NOTE: This function can only erase one flash sector at a time, and the other commands can be executed after the previous command has been completed.

Parameters

<i>config</i>	The pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
<i>key</i>	The value used to validate all flash erase APIs.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_-AlignmentError</i>	The parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_Address-Error</i>	The address is out of range.
<i>kStatus_FTFx_EraseKey-Error</i>	The API erase key is invalid.

### 13.2.5.4 status\_t FLASH\_EraseAll ( **flash\_config\_t \* config, uint32\_t key** )

Parameters

<i>config</i>	Pointer to the storage for the driver runtime state.
<i>key</i>	A value used to validate all flash erase APIs.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the all pflash and flexnvm were erased successfully, the swap and eeprom have been reset to unconfigured state.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_EraseKey-Error</i>	API erase key is invalid.

<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx-CommandFailure</i>	Run-time error during command execution.
<i>kStatus_FTFx_Partition-StatusUpdateFailure</i>	Failed to update the partition status.

### 13.2.5.5 **status\_t FLASH\_Program ( flash\_config\_t \* config, uint32\_t start, uint8\_t \* src, uint32\_t lengthInBytes )**

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be programmed. Must be word-aligned.
<i>src</i>	A pointer to the source buffer of data that is to be programmed into the flash.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the desired data were programmed successfully into flash based on desired start address and length.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx-AlignmentError</i>	Parameter is not aligned with the specified baseline.

<i>kStatus_FTFx_Address_Error</i>	Address is out of range.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

### 13.2.5.6 status\_t FLASH\_ProgramOnce ( *flash\_config\_t \* config, uint32\_t index, uint8\_t \* src, uint32\_t lengthInBytes* )

This function Program the Program-once-feild with given index and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>index</i>	The index indicating the area of program once field to be read.
<i>src</i>	A pointer to the source buffer of data that is used to store data to be write.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; The index indicating the area of program once field was programmed successfully.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.

<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

### 13.2.5.7 status\_t FLASH\_ReadResource ( *flash\_config\_t \* config, uint32\_t start, uint8\_t \* dst, uint32\_t lengthInBytes, ftfx\_read\_resource\_opt\_t option* )

This function reads the flash memory with the desired location for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be programmed. Must be word-aligned.
<i>dst</i>	A pointer to the destination buffer of data that is used to store data to be read.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be read. Must be word-aligned.
<i>option</i>	The resource option which indicates which area should be read back.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the data have been read successfully from program flash IFR, data flash IFR space, and the Version ID field.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.

<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

### 13.2.5.8 status\_t FLASH\_ReadOnce ( *flash\_config\_t \* config, uint32\_t index, uint8\_t \* dst, uint32\_t lengthInBytes* )

This function reads the read once feild with given index and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>index</i>	The index indicating the area of program once field to be read.
<i>dst</i>	A pointer to the destination buffer of data that is used to store data to be read.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the data have been successfully read form Program flash0 IFR map and Program Once field based on index and length.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

### 13.2.5.9 status\_t FLASH\_VerifyErase ( *flash\_config\_t \* config, uint32\_t start, uint32\_t lengthInBytes, ftfx\_margin\_value\_t margin* )

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

## Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
<i>margin</i>	Read margin choice.

## Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the specified FLASH region has been erased.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

**13.2.5.10 status\_t FLASH\_VerifyEraseAll ( flash\_config\_t \* *config*, ftfx\_margin\_value\_t *margin* )**

This function checks whether the flash is erased to the specified read margin level.

## Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>margin</i>	Read margin choice.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; all program flash and flexnvm were in erased state.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

### 13.2.5.11 **status\_t FLASH\_VerifyProgram ( flash\_config\_t \* *config*, uint32\_t *start*, uint32\_t *lengthInBytes*, const uint8\_t \* *expectedData*, ftfx\_margin\_value\_t *margin*, uint32\_t \* *failedAddress*, uint32\_t \* *failedData* )**

This function verifies the data programmed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be verified. Must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
<i>expectedData</i>	A pointer to the expected data that is to be verified against.
<i>margin</i>	Read margin choice.
<i>failedAddress</i>	A pointer to the returned failing address.
<i>failedData</i>	A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the desired data have been successfully programmed into specified FLASH region.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

### 13.2.5.12 status\_t FLASH\_GetSecurityState ( flash\_config\_t \* *config*, ftfx\_security\_state\_t \* *state* )

This function retrieves the current flash security status, including the security enabling state and the backdoor key enabling state.

Parameters

<i>config</i>	A pointer to storage for the driver runtime state.
<i>state</i>	A pointer to the value returned for the current security status code:

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the security state of flash was stored to state.
-----------------------------	---

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

### 13.2.5.13 status\_t FLASH\_SecurityBypass ( *flash\_config\_t \* config*, *const uint8\_t \* backdoorKey* )

If the MCU is in secured state, this function unsecures the MCU by comparing the provided backdoor key with ones in the flash configuration field.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>backdoorKey</i>	A pointer to the user buffer containing the backdoor key.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_-CommandFailure</i>	Run-time error during the command execution.

### 13.2.5.14 status\_t FLASH\_IsProtected ( *flash\_config\_t \* config*, *uint32\_t start*, *uint32\_t lengthInBytes*, *flash\_prot\_state\_t \* protection\_state* )

This function retrieves the current flash protect status for a given flash area as determined by the start address and length.

## Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be checked. Must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words) to be checked. Must be word-aligned.
<i>protection_state</i>	A pointer to the value returned for the current protection status code for the desired flash area.

## Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the protection state of specified FLASH region was stored to protection_state.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FTFx_AddressError</i>	The address is out of range.

**13.2.5.15 status\_t FLASH\_PflashSetProtection ( *flash\_config\_t \* config*, *pflash\_prot\_status\_t \* protectStatus* )**

## Parameters

<i>config</i>	A pointer to storage for the driver runtime state.
<i>protectStatus</i>	The expected protect status to set to the PFlash protection register. Each bit is corresponding to protection of 1/32(64) of the total PFlash. The least significant bit is corresponding to the lowest address area of PFlash. The most significant bit is corresponding to the highest address area of PFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

## Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the specified FLASH region is protected.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during command execution.

13.2.5.16 **status\_t FLASH\_PflashGetProtection ( flash\_config\_t \* *config*,  
pflash\_prot\_status\_t \* *protectStatus* )**

## Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>protectStatus</i>	Protect status returned by the PFlash IP. Each bit is corresponding to the protection of 1/32(64) of the total PFlash. The least significant bit corresponds to the lowest address area of the PFlash. The most significant bit corresponds to the highest address area of PFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

## Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the Protection state was stored to protect-Status;
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.

**13.2.5.17 status\_t FLASH\_GetProperty ( flash\_config\_t \* *config*, flash\_property\_tag\_t *whichProperty*, uint32\_t \* *value* )**

## Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>whichProperty</i>	The desired property from the list of properties in enum flash_property_tag_t
<i>value</i>	A pointer to the value returned for the desired flash property.

## Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the flash property was stored to value.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_Unknown-Property</i>	An unknown property tag.

**13.2.5.18 status\_t FLASH\_GetCommandState ( void )**

This function is used to obtain the execution status of the previous command.

Return values

<i>kStatus_FTFx_Success</i>	The previous command is executed successfully.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_-CommandFailure</i>	Run-time error during the command execution.

## 13.3 Fftfx CACHE Driver

### 13.3.1 Overview

#### Data Structures

- struct `fftfx_prefetch_speculation_status_t`  
*FTFx prefetch speculation status.* [More...](#)
- struct `fftfx_cache_config_t`  
*FTFx cache driver state information.* [More...](#)

#### Enumerations

- enum `_fftfx_cache_ram_func_constants` { `kFTFx_CACHE_RamFuncMaxSizeInWords` = 16U }  
*Constants for execute-in-RAM flash function.*

#### Functions

- `status_t FTFx_CACHE_Init (fftfx_cache_config_t *config)`  
*Initializes the global FTFx cache structure members.*
- `status_t FTFx_CACHE_ClearCachePrefetchSpeculation (fftfx_cache_config_t *config, bool isPreProcess)`  
*Process the cache/prefetch/speculation to the flash.*
- `status_t FTFx_CACHE_PflashSetPrefetchSpeculation (fftfx_prefetch_speculation_status_t *speculationStatus)`  
*Sets the PFlash prefetch speculation to the intended speculation status.*
- `status_t FTFx_CACHE_PflashGetPrefetchSpeculation (fftfx_prefetch_speculation_status_t *speculationStatus)`  
*Gets the PFlash prefetch speculation status.*

### 13.3.2 Data Structure Documentation

#### 13.3.2.1 struct fftfx\_prefetch\_speculation\_status\_t

##### Data Fields

- `bool instructionOff`  
*Instruction speculation.*
- `bool dataOff`  
*Data speculation.*

##### Field Documentation

(1) `bool fftfx_prefetch_speculation_status_t::instructionOff`

(2) `bool fftfx_prefetch_speculation_status_t::dataOff`

### 13.3.2.2 struct fftfx\_cache\_config\_t

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

#### Data Fields

- `uint8_t flashMemoryIndex`  
*0 - primary flash; 1 - secondary flash*
- `function_bit_operation_ptr_t bitOperFuncAddr`  
*An buffer point to the flash execute-in-RAM function.*

#### Field Documentation

(1) `function_bit_operation_ptr_t fftfx_cache_config_t::bitOperFuncAddr`

### 13.3.3 Enumeration Type Documentation

#### 13.3.3.1 enum \_fftfx\_cache\_ram\_func\_constants

Enumerator

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

### 13.3.4 Function Documentation

#### 13.3.4.1 status\_t FTFx\_CACHE\_Init( fftfx\_cache\_config\_t \* config )

This function checks and initializes the Flash module for the other FTFx cache APIs.

Parameters

<code>config</code>	Pointer to the storage for the driver runtime state.
---------------------	--

Return values

<code>kStatus_FTFx_Success</code>	API was executed successfully.
<code>kStatus_FTFx_Invalid-Argument</code>	An invalid argument is provided.

<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
---	---

### 13.3.4.2 status\_t FTFx\_CACHE\_ClearCachePrefetchSpeculation ( *ftfx\_cache\_config\_t \* config, bool isPreProcess* )

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>isPreProcess</i>	The possible option used to control flash cache/prefetch/speculation

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_Invalid-Argument</i>	Invalid argument is provided.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.

### 13.3.4.3 status\_t FTFx\_CACHE\_PflashSetPrefetchSpeculation ( *ftfx\_prefetch\_speculation\_status\_t \* speculationStatus* )

Parameters

<i>speculation-Status</i>	The expected protect status to set to the PFlash protection register. Each bit is
---------------------------	---

Return values

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

### 13.3.4.4 status\_t FTFx\_CACHE\_PflashGetPrefetchSpeculation ( *ftfx\_prefetch\_speculation\_status\_t \* speculationStatus* )

## Parameters

<i>speculation- Status</i>	Speculation status returned by the PFlash IP.
--------------------------------	---

## Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
-----------------------------	--------------------------------

## 13.4 Ftftx FLEXNVM Driver

### 13.4.1 Overview

#### Data Structures

- struct `flexnvm_config_t`  
*Flexnvm driver state information. [More...](#)*

#### Enumerations

- enum `flexnvm_property_tag_t` {
   
`kFLEXNVM_PropertyDflashSectorSize` = 0x00U,  
`kFLEXNVM_PropertyDflashTotalSize` = 0x01U,  
`kFLEXNVM_PropertyDflashBlockSize` = 0x02U,  
`kFLEXNVM_PropertyDflashBlockCount` = 0x03U,  
`kFLEXNVM_PropertyDflashBlockBaseAddr` = 0x04U,  
`kFLEXNVM_PropertyAliasDflashBlockBaseAddr` = 0x05U,  
`kFLEXNVM_PropertyFlexRamBlockBaseAddr` = 0x06U,  
`kFLEXNVM_PropertyFlexRamTotalSize` = 0x07U,  
`kFLEXNVM_PropertyEepromTotalSize` = 0x08U }

*Enumeration for various flexnvm properties.*

#### Functions

- `status_t FLEXNVM_EepromWrite (flexnvm_config_t *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)`  
*Programs the EEPROM with data at locations passed in through parameters.*

#### Initialization

- `status_t FLEXNVM_Init (flexnvm_config_t *config)`  
*Initializes the global flash properties structure members.*

#### Erasing

- `status_t FLEXNVM_DflashErase (flexnvm_config_t *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)`  
*Erases the Dflash sectors encompassed by parameters passed into function.*
- `status_t FLEXNVM_EraseAll (flexnvm_config_t *config, uint32_t key)`  
*Erases entire flexnvm.*

## Programming

- **status\_t FLEXNVM\_DflashProgram (flexnvm\_config\_t \*config, uint32\_t start, uint8\_t \*src, uint32\_t lengthInBytes)**  
*Programs flash with data at locations passed in through parameters.*
- **status\_t FLEXNVM\_ProgramPartition (flexnvm\_config\_t \*config, ftx\_partition\_flexram\_load\_opt\_t option, uint32\_t eepromDataSizeCode, uint32\_t flexnvmPartitionCode)**  
*Prepares the FlexNVM block for use as data flash, EEPROM backup, or a combination of both and initializes the FlexRAM.*

## Reading

- **status\_t FLEXNVM\_ReadResource (flexnvm\_config\_t \*config, uint32\_t start, uint8\_t \*dst, uint32\_t lengthInBytes, ftx\_read\_resource\_opt\_t option)**  
*Reads the resource with data at locations passed in through parameters.*

## Verification

- **status\_t FLEXNVM\_DflashVerifyErase (flexnvm\_config\_t \*config, uint32\_t start, uint32\_t lengthInBytes, ftx\_margin\_value\_t margin)**  
*Verifies an erasure of the desired flash area at a specified margin level.*
- **status\_t FLEXNVM\_VerifyEraseAll (flexnvm\_config\_t \*config, ftx\_margin\_value\_t margin)**  
*Verifies erasure of the entire flash at a specified margin level.*
- **status\_t FLEXNVM\_DflashVerifyProgram (flexnvm\_config\_t \*config, uint32\_t start, uint32\_t lengthInBytes, const uint8\_t \*expectedData, ftx\_margin\_value\_t margin, uint32\_t \*failedAddress, uint32\_t \*failedData)**  
*Verifies programming of the desired flash area at a specified margin level.*

## Security

- **status\_t FLEXNVM\_GetSecurityState (flexnvm\_config\_t \*config, ftx\_security\_state\_t \*state)**  
*Returns the security state via the pointer passed into the function.*
- **status\_t FLEXNVM\_SecurityBypass (flexnvm\_config\_t \*config, const uint8\_t \*backdoorKey)**  
*Allows users to bypass security with a backdoor key.*

## Flash Protection Utilities

- **status\_t FLEXNVM\_DflashSetProtection (flexnvm\_config\_t \*config, uint8\_t protectStatus)**  
*Sets the DFlash protection to the intended protection status.*
- **status\_t FLEXNVM\_DflashGetProtection (flexnvm\_config\_t \*config, uint8\_t \*protectStatus)**  
*Gets the DFlash protection status.*
- **status\_t FLEXNVM\_EepromSetProtection (flexnvm\_config\_t \*config, uint8\_t protectStatus)**  
*Sets the EEPROM protection to the intended protection status.*
- **status\_t FLEXNVM\_EepromGetProtection (flexnvm\_config\_t \*config, uint8\_t \*protectStatus)**

*Gets the EEPROM protection status.*

## Properties

- `status_t FLEXNVMGetProperty (flexnvm_config_t *config, flexnvm_property_tag_t whichProperty, uint32_t *value)`  
*Returns the desired flexnvm property.*

### 13.4.2 Data Structure Documentation

#### 13.4.2.1 struct flexnvm\_config\_t

An instance of this structure is allocated by the user of the Flexnvm driver and passed into each of the driver APIs.

### 13.4.3 Enumeration Type Documentation

#### 13.4.3.1 enum flexnvm\_property\_tag\_t

Enumerator

- `kFLEXNVM_PropertyDflashSectorSize` Dflash sector size property.
- `kFLEXNVM_PropertyDflashTotalSize` Dflash total size property.
- `kFLEXNVM_PropertyDflashBlockSize` Dflash block size property.
- `kFLEXNVM_PropertyDflashBlockCount` Dflash block count property.
- `kFLEXNVM_PropertyDflashBlockBaseAddr` Dflash block base address property.
- `kFLEXNVM_PropertyAliasDflashBlockBaseAddr` Dflash block base address Alias property.
- `kFLEXNVM_PropertyFlexRamBlockBaseAddr` FlexRam block base address property.
- `kFLEXNVM_PropertyFlexRamTotalSize` FlexRam total size property.
- `kFLEXNVM_PropertyEepromTotalSize` EEPROM total size property.

### 13.4.4 Function Documentation

#### 13.4.4.1 status\_t FLEXNVM\_Init ( `flexnvm_config_t * config` )

This function checks and initializes the Flash module for the other Flash APIs.

## Parameters

<i>config</i>	Pointer to the storage for the driver runtime state.
---------------	--

## Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_PartitionStatusUpdateFailure</i>	Failed to update the partition status.

**13.4.4.2 status\_t FLEXNVM\_DflashErase ( *flexnvm\_config\_t \* config, uint32\_t start, uint32\_t lengthInBytes, uint32\_t key* )**

This function erases the appropriate number of flash sectors based on the desired start address and length.

## Parameters

<i>config</i>	The pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words) to be erased. Must be word-aligned.
<i>key</i>	The value used to validate all flash erase APIs.

## Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the appropriate number of date flash sectors based on the desired start address and length were erased successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.

<i>kStatus_FTFx_AlignmentError</i>	The parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_AddressError</i>	The address is out of range.
<i>kStatus_FTFx_EraseKeyError</i>	The API erase key is invalid.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

### 13.4.4.3 status\_t FLEXNVM\_EraseAll ( *flexnvm\_config\_t \* config, uint32\_t key* )

Parameters

<i>config</i>	Pointer to the storage for the driver runtime state.
<i>key</i>	A value used to validate all flash erase APIs.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the entire flexnvm has been erased successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_EraseKeyError</i>	API erase key is invalid.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.

<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during command execution.
<i>kStatus_FTFx_PartitionStatusUpdateFailure</i>	Failed to update the partition status.

#### 13.4.4.4 **status\_t FLEXNVM\_DflashProgram ( flexnvm\_config\_t \* config, uint32\_t start, uint8\_t \* src, uint32\_t lengthInBytes )**

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be programmed. Must be word-aligned.
<i>src</i>	A pointer to the source buffer of data that is to be programmed into the flash.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the desired date have been successfully programed into specified date flash region.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.

<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_-CommandFailure</i>	Run-time error during the command execution.

#### 13.4.4.5 status\_t FLEXNVM\_ProgramPartition ( *flexnvm\_config\_t \* config, ftfx\_partition\_flexram\_load\_opt\_t option, uint32\_t eepromDataSizeCode, uint32\_t flexnvmPartitionCode* )

Parameters

<i>config</i>	Pointer to storage for the driver runtime state.
<i>option</i>	The option used to set FlexRAM load behavior during reset.
<i>eepromData-SizeCode</i>	Determines the amount of FlexRAM used in each of the available EEPROM subsystems.
<i>flexnvm-PartitionCode</i>	Specifies how to split the FlexNVM block between data flash memory and EEPROM backup memory supporting EEPROM functions.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the FlexNVM block for use as data flash, EEPROM backup, or a combination of both have been Prepared.
<i>kStatus_FTFx_Invalid-Argument</i>	Invalid argument is provided.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.

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

### 13.4.4.6 status\_t FLEXNVM\_ReadResource ( *flexnvm\_config\_t \* config*, *uint32\_t start*, *uint8\_t \* dst*, *uint32\_t lengthInBytes*, *ftfx\_read\_resource\_opt\_t option* )

This function reads the flash memory with the desired location for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be programmed. Must be word-aligned.
<i>dst</i>	A pointer to the destination buffer of data that is used to store data to be read.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be read. Must be word-aligned.
<i>option</i>	The resource option which indicates which area should be read back.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the data have been read successfully from program flash IFR, data flash IFR space, and the Version ID field
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_-AlignmentError</i>	Parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.

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

### 13.4.4.7 status\_t FLEXNVM\_DflashVerifyErase ( flexnvm\_config\_t \* config, uint32\_t start, uint32\_t lengthInBytes, ftfx\_margin\_value\_t margin )

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
<i>margin</i>	Read margin choice.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the specified data flash region is in erased state.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.

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

#### 13.4.4.8 status\_t FLEXNVM\_VerifyEraseAll ( *flexnvm\_config\_t \* config, ftfx\_margin\_value\_t margin* )

This function checks whether the flash is erased to the specified read margin level.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>margin</i>	Read margin choice.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the entire flexnvm region is in erased state.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

#### 13.4.4.9 status\_t FLEXNVM\_DflashVerifyProgram ( *flexnvm\_config\_t \* config, uint32\_t start, uint32\_t lengthInBytes, const uint8\_t \* expectedData, ftfx\_margin\_value\_t margin, uint32\_t \* failedAddress, uint32\_t \* failedData* )

This function verifies the data programmed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be verified. Must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
<i>expectedData</i>	A pointer to the expected data that is to be verified against.
<i>margin</i>	Read margin choice.
<i>failedAddress</i>	A pointer to the returned failing address.
<i>failedData</i>	A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the desired data have been programmed successfully into specified data flash region.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

#### 13.4.4.10 status\_t FLEXNVM\_GetSecurityState ( **flexnvm\_config\_t \* config,** **ftfx\_security\_state\_t \* state** )

This function retrieves the current flash security status, including the security enabling state and the backdoor key enabling state.

Parameters

<i>config</i>	A pointer to storage for the driver runtime state.
<i>state</i>	A pointer to the value returned for the current security status code:

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the security state of flexnvm was stored to state.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.

#### 13.4.4.11 **status\_t FLEXNVM\_SecurityBypass ( flexnvm\_config\_t \* *config*, const uint8\_t \* *backdoorKey* )**

If the MCU is in secured state, this function unsecures the MCU by comparing the provided backdoor key with ones in the flash configuration field.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>backdoorKey</i>	A pointer to the user buffer containing the backdoor key.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.

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

#### 13.4.4.12 status\_t FLEXNVM\_EepromWrite ( *flexnvm\_config\_t \* config, uint32\_t start, uint8\_t \* src, uint32\_t lengthInBytes* )

This function programs the emulated EEPROM with the desired data for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be programmed. Must be word-aligned.
<i>src</i>	A pointer to the source buffer of data that is to be programmed into the flash.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the desired data have been successfully programmed into specified eeprom region.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.
<i>kStatus_FTFx_SetFlexramAsEepromError</i>	Failed to set flexram as eeprom.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_RecoverFlexramAsRamError</i>	Failed to recover the FlexRAM as RAM.

#### 13.4.4.13 status\_t FLEXNVM\_DflashSetProtection ( *flexnvm\_config\_t \* config, uint8\_t protectStatus* )

## Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>protectStatus</i>	The expected protect status to set to the DFlash protection register. Each bit corresponds to the protection of the 1/8 of the total DFlash. The least significant bit corresponds to the lowest address area of the DFlash. The most significant bit corresponds to the highest address area of the DFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

## Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully; the specified DFlash region is protected.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_CommandNotSupported</i>	Flash API is not supported.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during command execution.

**13.4.4.14 status\_t FLEXNVM\_DflashGetProtection ( *flexnvm\_config\_t \* config, uint8\_t \* protectStatus* )**

## Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>protectStatus</i>	DFlash Protect status returned by the PFlash IP. Each bit corresponds to the protection of the 1/8 of the total DFlash. The least significant bit corresponds to the lowest address area of the DFlash. The most significant bit corresponds to the highest address area of the DFlash, and so on. There are two possible cases as below: 0: this area is protected. 1: this area is unprotected.

## Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.

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

### 13.4.4.15 status\_t FLEXNVM\_EepromSetProtection ( **flexnvm\_config\_t \* config, uint8\_t protectStatus** )

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>protectStatus</i>	The expected protect status to set to the EEPROM protection register. Each bit corresponds to the protection of the 1/8 of the total EEPROM. The least significant bit corresponds to the lowest address area of the EEPROM. The most significant bit corresponds to the highest address area of EEPROM, and so on. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_CommandNotSupported</i>	Flash API is not supported.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during command execution.

### 13.4.4.16 status\_t FLEXNVM\_EepromGetProtection ( **flexnvm\_config\_t \* config, uint8\_t \* protectStatus** )

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>protectStatus</i>	DFlash Protect status returned by the PFlash IP. Each bit corresponds to the protection of the 1/8 of the total EEPROM. The least significant bit corresponds to the lowest address area of the EEPROM. The most significant bit corresponds to the highest address area of the EEPROM. There are two possible cases as below: 0: this area is protected. 1: this area is unprotected.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_CommandNotSupported</i>	Flash API is not supported.

#### 13.4.4.17 **status\_t FLEXNVM\_GetProperty ( *flexnvm\_config\_t \* config,* *flexnvm\_property\_tag\_t whichProperty, uint32\_t \* value* )**

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>whichProperty</i>	The desired property from the list of properties in enum flexnvm_property_tag_t
<i>value</i>	A pointer to the value returned for the desired flexnvm property.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_UnknownProperty</i>	An unknown property tag.

## 13.5 ftx feature

### 13.5.1 Overview

#### Modules

- ftx adapter

#### Macros

- #define **FTFx\_DRIVER\_HAS\_FLASH1\_SUPPORT** (0U)  
*Indicates whether the secondary flash is supported in the Flash driver.*

#### FTFx configuration

- #define **FTFx\_DRIVER\_IS\_FLASH\_RESIDENT** 1U  
*Flash driver location.*
- #define **FTFx\_DRIVER\_IS\_EXPORTED** 0U  
*Flash Driver Export option.*

#### Secondary flash configuration

- #define **FTFx\_FLASH1\_HAS\_PROT\_CONTROL** (0U)  
*Indicates whether the secondary flash has its own protection register in flash module.*
- #define **FTFx\_FLASH1\_HAS\_XACC\_CONTROL** (0U)  
*Indicates whether the secondary flash has its own Execute-Only access register in flash module.*

### 13.5.2 Macro Definition Documentation

#### 13.5.2.1 #define FTFx\_DRIVER\_IS\_FLASH\_RESIDENT 1U

Used for the flash resident application.

#### 13.5.2.2 #define FTFx\_DRIVER\_IS\_EXPORTED 0U

Used for the MCUXpresso SDK application.

#### 13.5.2.3 #define FTFx\_FLASH1\_HAS\_PROT\_CONTROL (0U)

#### 13.5.2.4 #define FTFx\_FLASH1\_HAS\_XACC\_CONTROL (0U)

### **13.5.3 ftx adapter**

## 13.6 ftx controller

### 13.6.1 Overview

#### Modules

- [ftfx utilities](#)

#### Data Structures

- struct [ftfx\\_spec\\_mem\\_t](#)  
*ftfx special memory access information.* [More...](#)
- struct [ftfx\\_mem\\_desc\\_t](#)  
*Flash memory descriptor.* [More...](#)
- struct [ftfx\\_ops\\_config\\_t](#)  
*Active FTFx information for the current operation.* [More...](#)
- struct [ftfx\\_ifr\\_desc\\_t](#)  
*Flash IFR memory descriptor.* [More...](#)
- struct [ftfx\\_config\\_t](#)  
*Flash driver state information.* [More...](#)

#### Enumerations

- enum [ftfx\\_partition\\_flexram\\_load\\_opt\\_t](#) {
   
kFTFx\_PartitionFlexramLoadOptLoadedWithValidEepromData,
   
kFTFx\_PartitionFlexramLoadOptNotLoaded = 0x01U }
   
*Enumeration for the FlexRAM load during reset option.*
- enum [ftfx\\_read\\_resource\\_opt\\_t](#) {
   
kFTFx\_ResourceOptionFlashIfr,
   
kFTFx\_ResourceOptionVersionId = 0x01U }
   
*Enumeration for the two possible options of flash read resource command.*
- enum [ftfx\\_margin\\_value\\_t](#) {
   
kFTFx\_MarginValueNormal,
   
kFTFx\_MarginValueUser,
   
kFTFx\_MarginValueFactory,
   
kFTFx\_MarginValueInvalid }
   
*Enumeration for supported FTFx margin levels.*
- enum [ftfx\\_security\\_state\\_t](#) {
   
kFTFx\_SecurityStateNotSecure = (int)0xc33cc33cu,
   
kFTFx\_SecurityStateBackdoorEnabled = (int)0x5aa55aa5u,
   
kFTFx\_SecurityStateBackdoorDisabled = (int)0x5ac33ca5u }
   
*Enumeration for the three possible FTFx security states.*
- enum [ftfx\\_flexram\\_func\\_opt\\_t](#) {
   
kFTFx\_FlexramFuncOptAvailableAsRam = 0xFFU,
   
kFTFx\_FlexramFuncOptAvailableForEeprom = 0x00U }
   
*Enumeration for the two possilbe options of set FlexRAM function command.*

- enum `_flash_acceleration_ram_property`  
*Enumeration for acceleration ram property.*
- enum `ftfx_swap_state_t` {
   
`kFTFx_SwapStateUninitialized` = 0x00U,  
`kFTFx_SwapStateReady` = 0x01U,  
`kFTFx_SwapStateUpdate` = 0x02U,  
`kFTFx_SwapStateUpdateErased` = 0x03U,  
`kFTFx_SwapStateComplete` = 0x04U,  
`kFTFx_SwapStateDisabled` = 0x05U }
- Enumeration for the possible flash Swap status.*
- enum `_ftfx_memory_type`  
*Enumeration for FTFx memory type.*

## FTFx status

- enum {
   
`kStatus_FTFx_Success` = MAKE\_STATUS(kStatusGroupGeneric, 0),  
`kStatus_FTFx_InvalidArgument` = MAKE\_STATUS(kStatusGroupGeneric, 4),  
`kStatus_FTFx_SizeError` = MAKE\_STATUS(kStatusGroupFtxDriver, 0),  
`kStatus_FTFx_AlignmentError`,  
`kStatus_FTFx_AddressError` = MAKE\_STATUS(kStatusGroupFtxDriver, 2),  
`kStatus_FTFx_AccessError`,  
`kStatus_FTFx_ProtectionViolation`,  
`kStatus_FTFx_CommandFailure`,  
`kStatus_FTFx_UnknownProperty` = MAKE\_STATUS(kStatusGroupFtxDriver, 6),  
`kStatus_FTFx_EraseKeyError` = MAKE\_STATUS(kStatusGroupFtxDriver, 7),  
`kStatus_FTFx_RegionExecuteOnly` = MAKE\_STATUS(kStatusGroupFtxDriver, 8),  
`kStatus_FTFx_ExecuteInRamFunctionNotReady`,  
`kStatus_FTFx_PartitionStatusUpdateFailure`,  
`kStatus_FTFx_SetFlexramAsEepromError`,  
`kStatus_FTFx_RecoverFlexramAsRamError`,  
`kStatus_FTFx_SetFlexramAsRamError` = MAKE\_STATUS(kStatusGroupFtxDriver, 13),  
`kStatus_FTFx_RecoverFlexramAsEepromError`,  
`kStatus_FTFx_CommandNotSupported` = MAKE\_STATUS(kStatusGroupFtxDriver, 15),  
`kStatus_FTFx_SwapSystemNotInUninitialized`,  
`kStatus_FTFx_SwapIndicatorAddressError`,  
`kStatus_FTFx_ReadOnlyProperty` = MAKE\_STATUS(kStatusGroupFtxDriver, 18),  
`kStatus_FTFx_InvalidPropertyValue`,  
`kStatus_FTFx_InvalidSpeculationOption`,  
`kStatus_FTFx_CommandOperationInProgress` }
- FTFx driver status codes.*
- #define `kStatusGroupGeneric` 0  
*FTFx driver status group.*
- #define `kStatusGroupFtxDriver` 1

## FTFx API key

- enum `_ftfx_driver_api_keys` { `kFTFx_ApiEraseKey` = FOUR\_CHAR\_CODE('k', 'f', 'e', 'k') }
- Enumeration for FTFx driver API keys.*

## Initialization

- void `FTFx_API_Init (ftfx_config_t *config)`  
*Initializes the global flash properties structure members.*

## Erasing

- `status_t FTFx_CMD_Erase (ftfx_config_t *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)`  
*Erases the flash sectors encompassed by parameters passed into function.*
- `status_t FTFx_CMD_EraseSectorNonBlocking (ftfx_config_t *config, uint32_t start, uint32_t key)`  
*Erases the flash sectors encompassed by parameters passed into function.*
- `status_t FTFx_CMD_EraseAll (ftfx_config_t *config, uint32_t key)`  
*Erases entire flash.*
- `status_t FTFx_CMD_EraseAllExecuteOnlySegments (ftfx_config_t *config, uint32_t key)`  
*Erases all program flash execute-only segments defined by the FXACC registers.*

## Programming

- `status_t FTFx_CMD_Program (ftfx_config_t *config, uint32_t start, const uint8_t *src, uint32_t lengthInBytes)`  
*Programs flash with data at locations passed in through parameters.*
- `status_t FTFx_CMD_ProgramOnce (ftfx_config_t *config, uint32_t index, const uint8_t *src, uint32_t lengthInBytes)`  
*Programs Program Once Field through parameters.*

## Reading

- `status_t FTFx_CMD_ReadOnce (ftfx_config_t *config, uint32_t index, uint8_t *dst, uint32_t lengthInBytes)`  
*Reads the Program Once Field through parameters.*
- `status_t FTFx_CMD_ReadResource (ftfx_config_t *config, uint32_t start, uint8_t *dst, uint32_t lengthInBytes, ftfx_read_resource_opt_t option)`  
*Reads the resource with data at locations passed in through parameters.*

## Verification

- `status_t FTFx_CMD_VerifyErase (ftfx_config_t *config, uint32_t start, uint32_t lengthInBytes, ftx_margin_value_t margin)`  
*Verifies an erasure of the desired flash area at a specified margin level.*
- `status_t FTFx_CMD_VerifyEraseAll (ftfx_config_t *config, ftx_margin_value_t margin)`  
*Verifies erasure of the entire flash at a specified margin level.*
- `status_t FTFx_CMD_VerifyEraseAllExecuteOnlySegments (ftfx_config_t *config, ftx_margin_value_t margin)`  
*Verifies whether the program flash execute-only segments have been erased to the specified read margin level.*
- `status_t FTFx_CMD_VerifyProgram (ftfx_config_t *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, ftx_margin_value_t margin, uint32_t *failedAddress, uint32_t *failedData)`  
*Verifies programming of the desired flash area at a specified margin level.*

## Security

- `status_t FTFx_REG_GetSecurityState (ftfx_config_t *config, ftx_security_state_t *state)`  
*Returns the security state via the pointer passed into the function.*
- `status_t FTFx_CMD_SecurityBypass (ftfx_config_t *config, const uint8_t *backdoorKey)`  
*Allows users to bypass security with a backdoor key.*

### 13.6.2 Data Structure Documentation

#### 13.6.2.1 struct ftx\_spec\_mem\_t

##### Data Fields

- `uint32_t base`  
*Base address of flash special memory.*
- `uint32_t size`  
*size of flash special memory.*
- `uint32_t count`  
*flash special memory count.*

##### Field Documentation

- (1) `uint32_t ftx_spec_mem_t::base`
- (2) `uint32_t ftx_spec_mem_t::size`
- (3) `uint32_t ftx_spec_mem_t::count`

### 13.6.2.2 struct ftfx\_mem\_desc\_t

#### Data Fields

- `uint32_t blockBase`  
*A base address of the flash block.*
- `uint32_t totalSize`  
*The size of the flash block.*
- `uint32_t sectorSize`  
*The size in bytes of a sector of flash.*
- `uint32_t blockCount`  
*A number of flash blocks.*
- `uint8_t type`  
*Type of flash block.*
- `uint8_t index`  
*Index of flash block.*

#### Field Documentation

- (1) `uint8_t ftfx_mem_desc_t::type`
- (2) `uint8_t ftfx_mem_desc_t::index`
- (3) `uint32_t ftfx_mem_desc_t::totalSize`
- (4) `uint32_t ftfx_mem_desc_t::sectorSize`
- (5) `uint32_t ftfx_mem_desc_t::blockCount`

### 13.6.2.3 struct ftfx\_ops\_config\_t

#### Data Fields

- `uint32_t convertedAddress`  
*A converted address for the current flash type.*

#### Field Documentation

- (1) `uint32_t ftfx_ops_config_t::convertedAddress`

### 13.6.2.4 struct ftfx\_ifr\_desc\_t

### 13.6.2.5 struct ftfx\_config\_t

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

## Data Fields

- `uint32_t flexramBlockBase`  
*The base address of the FlexRAM/acceleration RAM.*
- `uint32_t flexramTotalSize`  
*The size of the FlexRAM/acceleration RAM.*
- `uint16_t eepromTotalSize`  
*The size of EEPROM area which was partitioned from FlexRAM.*
- `function_ptr_t runCmdFuncAddr`  
*An buffer point to the flash execute-in-RAM function.*

### Field Documentation

(1) `function_ptr_t ftfx_config_t::runCmdFuncAddr`

## 13.6.3 Macro Definition Documentation

### 13.6.3.1 #define kStatusGroupGeneric 0

## 13.6.4 Enumeration Type Documentation

### 13.6.4.1 anonymous enum

Enumerator

`kStatus_FTFx_Success` API is executed successfully.

`kStatus_FTFx_InvalidArgument` Invalid argument.

`kStatus_FTFx_SizeError` Error size.

`kStatus_FTFx_AlignmentError` Parameter is not aligned with the specified baseline.

`kStatus_FTFx_AddressError` Address is out of range.

`kStatus_FTFx_AccessError` Invalid instruction codes and out-of bound addresses.

`kStatus_FTFx_ProtectionViolation` The program/erase operation is requested to execute on protected areas.

`kStatus_FTFx_CommandFailure` Run-time error during command execution.

`kStatus_FTFx_UnknownProperty` Unknown property.

`kStatus_FTFx_EraseKeyError` API erase key is invalid.

`kStatus_FTFx_RegionExecuteOnly` The current region is execute-only.

`kStatus_FTFx_ExecuteInRamFunctionNotReady` Execute-in-RAM function is not available.

`kStatus_FTFx_PartitionStatusUpdateFailure` Failed to update partition status.

`kStatus_FTFx_SetFlexramAsEepromError` Failed to set FlexRAM as EEPROM.

`kStatus_FTFx_RecoverFlexramAsRamError` Failed to recover FlexRAM as RAM.

`kStatus_FTFx_SetFlexramAsRamError` Failed to set FlexRAM as RAM.

`kStatus_FTFx_RecoverFlexramAsEepromError` Failed to recover FlexRAM as EEPROM.

`kStatus_FTFx_CommandNotSupported` Flash API is not supported.

`kStatus_FTFx_SwapSystemNotInUninitialized` Swap system is not in an uninitialized state.

`kStatus_FTFx_SwapIndicatorAddressError` The swap indicator address is invalid.

`kStatus_FTFx_ReadOnlyProperty` The flash property is read-only.

*kStatus\_FTFx\_InvalidPropertyValue* The flash property value is out of range.

*kStatus\_FTFx\_InvalidSpeculationOption* The option of flash prefetch speculation is invalid.

*kStatus\_FTFx\_CommandOperationInProgress* The option of flash command is processing.

#### 13.6.4.2 enum \_ftfx\_driver\_api\_keys

Note

The resulting value is built with a byte order such that the string being readable in expected order when viewed in a hex editor, if the value is treated as a 32-bit little endian value.

Enumerator

*kFTFx\_ApiEraseKey* Key value used to validate all FTFx erase APIs.

#### 13.6.4.3 enum ftfx\_partition\_flexram\_load\_opt\_t

Enumerator

*kFTFx\_PartitionFlexramLoadOptLoadedWithValidEepromData* FlexRAM is loaded with valid EEPROM data during reset sequence.

*kFTFx\_PartitionFlexramLoadOptNotLoaded* FlexRAM is not loaded during reset sequence.

#### 13.6.4.4 enum ftfx\_read\_resource\_opt\_t

Enumerator

*kFTFx\_ResourceOptionFlashIfr* Select code for Program flash 0 IFR, Program flash swap 0 IFR, Data flash 0 IFR.

*kFTFx\_ResourceOptionVersionId* Select code for the version ID.

#### 13.6.4.5 enum ftfx\_margin\_value\_t

Enumerator

*kFTFx\_MarginValueNormal* Use the 'normal' read level for 1s.

*kFTFx\_MarginValueUser* Apply the 'User' margin to the normal read-1 level.

*kFTFx\_MarginValueFactory* Apply the 'Factory' margin to the normal read-1 level.

*kFTFx\_MarginValueInvalid* Not real margin level, Used to determine the range of valid margin level.

### 13.6.4.6 enum ftfx\_security\_state\_t

Enumerator

*kFTFx\_SecurityStateNotSecure* Flash is not secure.

*kFTFx\_SecurityStateBackdoorEnabled* Flash backdoor is enabled.

*kFTFx\_SecurityStateBackdoorDisabled* Flash backdoor is disabled.

### 13.6.4.7 enum ftfx\_flexram\_func\_opt\_t

Enumerator

*kFTFx\_FlexramFuncOptAvailableAsRam* An option used to make FlexRAM available as RAM.

*kFTFx\_FlexramFuncOptAvailableForEeprom* An option used to make FlexRAM available for E-EPROM.

### 13.6.4.8 enum ftfx\_swap\_state\_t

Enumerator

*kFTFx\_SwapStateUninitialized* Flash Swap system is in an uninitialized state.

*kFTFx\_SwapStateReady* Flash Swap system is in a ready state.

*kFTFx\_SwapStateUpdate* Flash Swap system is in an update state.

*kFTFx\_SwapStateUpdateErased* Flash Swap system is in an updateErased state.

*kFTFx\_SwapStateComplete* Flash Swap system is in a complete state.

*kFTFx\_SwapStateDisabled* Flash Swap system is in a disabled state.

## 13.6.5 Function Documentation

### 13.6.5.1 void FTFx\_API\_Init ( ftfx\_config\_t \* config )

This function checks and initializes the Flash module for the other Flash APIs.

Parameters

<i>config</i>	Pointer to the storage for the driver runtime state.
---------------	--

### 13.6.5.2 status\_t FTFx\_CMD\_Erase ( ftfx\_config\_t \* config, uint32\_t start, uint32\_t lengthInBytes, uint32\_t key )

This function erases the appropriate number of flash sectors based on the desired start address and length.

## Parameters

<i>config</i>	The pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words) to be erased. Must be word-aligned.
<i>key</i>	The value used to validate all flash erase APIs.

## Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	The parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_AddressError</i>	The address is out of range.
<i>kStatus_FTFx_EraseKeyError</i>	The API erase key is invalid.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

**13.6.5.3 status\_t FTFx\_CMD\_EraseSectorNonBlocking ( *ftfx\_config\_t \* config, uint32\_t start, uint32\_t key* )**

This function erases one flash sector size based on the start address.

## Parameters

<i>config</i>	The pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
<i>key</i>	The value used to validate all flash erase APIs.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	The parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_AddressError</i>	The address is out of range.
<i>kStatus_FTFx_EraseKeyError</i>	The API erase key is invalid.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.

#### 13.6.5.4 status\_t FTFx\_CMD\_EraseAll ( *ftfx\_config\_t \* config, uint32\_t key* )

Parameters

<i>config</i>	Pointer to the storage for the driver runtime state.
<i>key</i>	A value used to validate all flash erase APIs.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_EraseKeyError</i>	API erase key is invalid.

<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_-CommandFailure</i>	Run-time error during command execution.
<i>kStatus_FTFx_Partition-StatusUpdateFailure</i>	Failed to update the partition status.

### 13.6.5.5 **status\_t FTFx\_CMD\_EraseAllExecuteOnlySegments ( ftfx\_config\_t \* config, uint32\_t key )**

Parameters

<i>config</i>	Pointer to the storage for the driver runtime state.
<i>key</i>	A value used to validate all flash erase APIs.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_EraseKey-Error</i>	API erase key is invalid.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.

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

### 13.6.5.6 **status\_t FTFx\_CMD\_Program ( ftfx\_config\_t \* config, uint32\_t start, const uint8\_t \* src, uint32\_t lengthInBytes )**

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be programmed. Must be word-aligned.
<i>src</i>	A pointer to the source buffer of data that is to be programmed into the flash.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.

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

### 13.6.5.7 status\_t FTFx\_CMD\_ProgramOnce ( *ftfx\_config\_t \* config, uint32\_t index, const uint8\_t \* src, uint32\_t lengthInBytes* )

This function programs the Program Once Field with the desired data for a given flash area as determined by the index and length.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>index</i>	The index indicating which area of the Program Once Field to be programmed.
<i>src</i>	A pointer to the source buffer of data that is to be programmed into the Program Once Field.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

### 13.6.5.8 status\_t FTFx\_CMD\_ReadOnce ( *ftfx\_config\_t \* config, uint32\_t index, uint8\_t \* dst, uint32\_t lengthInBytes* )

This function reads the read once feild with given index and length.

## Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>index</i>	The index indicating the area of program once field to be read.
<i>dst</i>	A pointer to the destination buffer of data that is used to store data to be read.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

## Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_-CommandFailure</i>	Run-time error during the command execution.

### 13.6.5.9 status\_t FTFx\_CMD\_ReadResource ( *ftfx\_config\_t \* config*, *uint32\_t start*, *uint8\_t \* dst*, *uint32\_t lengthInBytes*, *ftfx\_read\_resource\_opt\_t option* )

This function reads the flash memory with the desired location for a given flash area as determined by the start address and length.

## Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be programmed. Must be word-aligned.
<i>dst</i>	A pointer to the destination buffer of data that is used to store data to be read.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be read. Must be word-aligned.

<i>option</i>	The resource option which indicates which area should be read back.
---------------	---

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with the specified baseline.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

### 13.6.5.10 `status_t FTFx_CMD_VerifyErase( ftfx_config_t * config, uint32_t start, uint32_t lengthInBytes, ftfx_margin_value_t margin )`

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
<i>margin</i>	Read margin choice.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

### 13.6.5.11 status\_t FTFx\_CMD\_VerifyEraseAll( ftfx\_config\_t \* config, ftfx\_margin\_value\_t margin )

This function checks whether the flash is erased to the specified read margin level.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>margin</i>	Read margin choice.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.

<i>kStatus_FTFx_Access_Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

### 13.6.5.12 status\_t FTFx\_CMD\_VerifyEraseAllExecuteOnlySegments ( *ftfx\_config\_t \* config, ftfx\_margin\_value\_t margin* )

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>margin</i>	Read margin choice.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access_Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

### 13.6.5.13 status\_t FTFx\_CMD\_VerifyProgram ( *ftfx\_config\_t \* config, uint32\_t start, uint32\_t lengthInBytes, const uint8\_t \* expectedData, ftfx\_margin\_value\_t margin, uint32\_t \* failedAddress, uint32\_t \* failedData* )

This function verifies the data programed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

## Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be verified. Must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
<i>expectedData</i>	A pointer to the expected data that is to be verified against.
<i>margin</i>	Read margin choice.
<i>failedAddress</i>	A pointer to the returned failing address.
<i>failedData</i>	A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

## Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_InvalidArgument</i>	An invalid argument is provided.
<i>kStatus_FTFx_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FTFx_AddressError</i>	Address is out of range.
<i>kStatus_FTFx_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_CommandFailure</i>	Run-time error during the command execution.

### 13.6.5.14 **status\_t FTFx\_REG\_GetSecurityState ( ftfx\_config\_t \* config, ftfx\_security\_state\_t \* state )**

This function retrieves the current flash security status, including the security enabling state and the backdoor key enabling state.

Parameters

<i>config</i>	A pointer to storage for the driver runtime state.
<i>state</i>	A pointer to the value returned for the current security status code:

Return values

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

### 13.6.5.15 **status\_t FTFx\_CMD\_SecurityBypass ( *ftfx\_config\_t \* config, const uint8\_t \* backdoorKey* )**

If the MCU is in secured state, this function unsecures the MCU by comparing the provided backdoor key with ones in the flash configuration field.

Parameters

<i>config</i>	A pointer to the storage for the driver runtime state.
<i>backdoorKey</i>	A pointer to the user buffer containing the backdoor key.

Return values

<i>kStatus_FTFx_Success</i>	API was executed successfully.
<i>kStatus_FTFx_Invalid-Argument</i>	An invalid argument is provided.
<i>kStatus_FTFx_ExecuteIn-RamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FTFx_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FTFx_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FTFx_-CommandFailure</i>	Run-time error during the command execution.

## 13.6.6 ftx utilities

### 13.6.6.1 Overview

#### Macros

- `#define MAKE_VERSION(major, minor, bugfix) (((major) << 16) | ((minor) << 8) | (bugfix))`  
*Constructs the version number for drivers.*
- `#define MAKE_STATUS(group, code) (((group)*100) + (code)))`  
*Constructs a status code value from a group and a code number.*
- `#define FOUR_CHAR_CODE(a, b, c, d) (((uint32_t)(d) << 24u) | ((uint32_t)(c) << 16u) | ((uint32_t)(b) << 8u) | ((uint32_t)(a)))`  
*Constructs the four character code for the Flash driver API key.*
- `#define B1P4(b) (((uint32_t)(b)&0xFFU) << 24U)`  
*bytes2word utility.*

#### Alignment macros

- `#define ALIGN_DOWN(x, a) (((uint32_t)(x)) & ~((uint32_t)(a)-1u))`  
*Alignment(down) utility.*
- `#define ALIGN_UP(x, a) ALIGN_DOWN((uint32_t)(x) + (uint32_t)(a)-1u, a)`  
*Alignment(up) utility.*

### 13.6.6.2 Macro Definition Documentation

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

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

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

**13.6.6.2.4 `#define ALIGN_DOWN( x, a ) (((uint32_t)(x)) & ~((uint32_t)(a)-1u))`**

**13.6.6.2.5 `#define ALIGN_UP( x, a ) ALIGN_DOWN((uint32_t)(x) + (uint32_t)(a)-1u, a)`**

**13.6.6.2.6 `#define B1P4( b ) (((uint32_t)(b)&0xFFU) << 24U)`**

# Chapter 14

## GPIO: General-Purpose Input/Output Driver

### 14.1 Overview

#### Modules

- FGPIO Driver
- GPIO Driver

#### Data Structures

- struct `gpio_pin_config_t`  
*The GPIO pin configuration structure. [More...](#)*

#### Macros

- #define `GPIO_FIT_REG`(value) ((`uint8_t`)(value))  
*For some platforms with 8-bit register width, cast the type to `uint8_t`.*

#### Enumerations

- enum `gpio_pin_direction_t` {  
  `kGPIO_DigitalInput` = 0U,  
  `kGPIO_DigitalOutput` = 1U }  
*GPIO direction definition.*
- enum `gpio_checker_attribute_t` {  
  `kGPIO_UsernonsecureRWUsersecureRWPrivilegedsecureRW`,  
  `kGPIO_UsernonsecureRUsersecureRWPrivilegedsecureRW`,  
  `kGPIO_UsernonsecureNUsersecureRWPrivilegedsecureRW`,  
  `kGPIO_UsernonsecureRUsersecureRPrivilegedsecureRW`,  
  `kGPIO_UsernonsecureNUsersecureRPrivilegedsecureRW`,  
  `kGPIO_UsernonsecureNUsersecureNPrivilegedsecureRW`,  
  `kGPIO_UsernonsecureNUsersecureNPrivilegedsecureR`,  
  `kGPIO_UsernonsecureNUsersecureNPrivilegedsecureN`,  
  `kGPIO_IgnoreAttributeCheck` = 0x80U }  
*GPIO checker attribute.*

#### Driver version

- #define `FSL_GPIO_DRIVER_VERSION` (`MAKE_VERSION`(2, 6, 0))  
*GPIO driver version.*

### 14.2 Data Structure Documentation

### 14.2.1 struct gpio\_pin\_config\_t

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the [PORT\\_SetPinConfig\(\)](#).

#### Data Fields

- `gpio_pin_direction_t pinDirection`  
*GPIO direction, input or output.*
- `uint8_t outputLogic`  
*Set a default output logic, which has no use in input.*

### 14.3 Macro Definition Documentation

#### 14.3.1 #define FSL\_GPIO\_DRIVER\_VERSION (MAKE\_VERSION(2, 6, 0))

### 14.4 Enumeration Type Documentation

#### 14.4.1 enum gpio\_pin\_direction\_t

Enumerator

*kGPIO\_DigitalInput* Set current pin as digital input.

*kGPIO\_DigitalOutput* Set current pin as digital output.

#### 14.4.2 enum gpio\_checker\_attribute\_t

Enumerator

*kGPIO\_UsernonsecureRWUsersecureRWPrivilegedsecureRW* User nonsecure:Read+Write; User Secure:Read+Write; Privileged Secure:Read+Write.

*kGPIO\_UsernonsecureRUsersecureRWPrivilegedsecureRW* User nonsecure:Read; User Secure:Read+Write; Privileged Secure:Read+Write.

*kGPIO\_UsernonsecureNUsersecureRWPrivilegedsecureRW* User nonsecure:None; User Secure:Read+Write; Privileged Secure:Read+Write.

*kGPIO\_UsernonsecureRUsersecureRPrivilegedsecureRW* User nonsecure:Read; User Secure:Read; Privileged Secure:Read+Write.

*kGPIO\_UsernonsecureNUsersecureRPrivilegedsecureRW* User nonsecure:None; User Secure:Read; Privileged Secure:Read+Write.

*kGPIO\_UsernonsecureNUsersecureNPrivilegedsecureRW* User nonsecure:None; User Secure:None; Privileged Secure:Read+Write.

*kGPIO\_UsernonsecureNUsersecureNPrivilegedsecureR* User nonsecure:None; User Secure:None; Privileged Secure:Read.

***kGPIO\_UsernonsecureNUsersecureNPrivilegedsecureN*** User nonsecure:None; User Secure-  
:None; Privileged Secure:None.

***kGPIO\_IgnoreAttributeCheck*** Ignores the attribute check.

## 14.5 GPIO Driver

### 14.5.1 Overview

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

### 14.5.2 Typical use case

#### 14.5.2.1 Output Operation

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

#### 14.5.2.2 Input Operation

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

## GPIO Configuration

- void [GPIO\\_PinInit](#) (GPIO\_Type \*base, uint32\_t pin, const [gpio\\_pin\\_config\\_t](#) \*config)  
*Initializes a GPIO pin used by the board.*

## GPIO Output Operations

- static void [GPIO\\_PinWrite](#) (GPIO\_Type \*base, uint32\_t pin, uint8\_t output)  
*Sets the output level of the multiple GPIO pins to the logic 1 or 0.*
- static void [GPIO\\_PortSet](#) (GPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 1.*
- static void [GPIO\\_PortClear](#) (GPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 0.*
- static void [GPIO\\_PortToggle](#) (GPIO\_Type \*base, uint32\_t mask)  
*Reverses the current output logic of the multiple GPIO pins.*

## GPIO Input Operations

- static uint32\_t [GPIO\\_PinRead](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current input value of the GPIO port.*

## GPIO Interrupt

- uint32\_t [GPIO\\_PortGetInterruptFlags](#) (GPIO\_Type \*base)  
*Reads the GPIO port interrupt status flag.*

- void [GPIO\\_PortClearInterruptFlags](#) (GPIO\_Type \*base, uint32\_t mask)  
*Clears multiple GPIO pin interrupt status flags.*
- void [GPIO\\_CheckAttributeBytes](#) (GPIO\_Type \*base, [gpio\\_checker\\_attribute\\_t](#) attribute)  
*brief The GPIO module supports a device-specific number of data ports, organized as 32-bit words/8-bit Bytes.*

### 14.5.3 Function Documentation

#### 14.5.3.1 void [GPIO\\_PinInit](#) ( **GPIO\_Type** \* *base*, **uint32\_t** *pin*, **const gpio\_pin\_config\_t** \* *config* )

To initialize the GPIO, define a pin configuration, as either input or output, in the user file. Then, call the [GPIO\\_PinInit\(\)](#) function.

This is an example to define an input pin or an output pin configuration.

```
* Define a digital input pin configuration,
* gpio\_pin\_config\_t config =
* {
*   kGPIO\_DigitalInput,
*   0,
* }
* Define a digital output pin configuration,
* gpio\_pin\_config\_t config =
* {
*   kGPIO\_DigitalOutput,
*   0,
* }
```

Parameters

<i>base</i>	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
<i>pin</i>	GPIO port pin number
<i>config</i>	GPIO pin configuration pointer

#### 14.5.3.2 static void [GPIO\\_PinWrite](#) ( **GPIO\_Type** \* *base*, **uint32\_t** *pin*, **uint8\_t** *output* ) [\[inline\]](#), [\[static\]](#)

Parameters

<i>base</i>	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
<i>pin</i>	GPIO pin number
<i>output</i>	<p>GPIO pin output logic level.</p> <ul style="list-style-type: none"> <li>• 0: corresponding pin output low-logic level.</li> <li>• 1: corresponding pin output high-logic level.</li> </ul>

#### 14.5.3.3 static void GPIO\_PortSet ( **GPIO\_Type** \* *base*, **uint32\_t** *mask* ) [inline], [static]

Parameters

<i>base</i>	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
<i>mask</i>	GPIO pin number macro

#### 14.5.3.4 static void GPIO\_PortClear ( **GPIO\_Type** \* *base*, **uint32\_t** *mask* ) [inline], [static]

Parameters

<i>base</i>	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
<i>mask</i>	GPIO pin number macro

#### 14.5.3.5 static void GPIO\_PortToggle ( **GPIO\_Type** \* *base*, **uint32\_t** *mask* ) [inline], [static]

Parameters

<i>base</i>	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
<i>mask</i>	GPIO pin number macro

#### 14.5.3.6 static **uint32\_t** GPIO\_PinRead ( **GPIO\_Type** \* *base*, **uint32\_t** *pin* ) [inline], [static]

Parameters

<i>base</i>	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
<i>pin</i>	GPIO pin number

Return values

<i>GPIO</i>	port input value <ul style="list-style-type: none"> <li>• 0: corresponding pin input low-logic level.</li> <li>• 1: corresponding pin input high-logic level.</li> </ul>
-------------	--

#### 14.5.3.7 `uint32_t GPIO_PortGetInterruptFlags ( GPIO_Type * base )`

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

<i>base</i>	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
-------------	--

Return values

<i>The</i>	current GPIO port interrupt status flag, for example, 0x00010001 means the pin 0 and 17 have the interrupt.
------------	---

#### 14.5.3.8 `void GPIO_PortClearInterruptFlags ( GPIO_Type * base, uint32_t mask )`

Parameters

<i>base</i>	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
<i>mask</i>	GPIO pin number macro

#### 14.5.3.9 `void GPIO_CheckAttributeBytes ( GPIO_Type * base, gpio_checker_attribute_t attribute )`

Each 32-bit/8-bit data port includes a GACR register, which defines the byte-level attributes required for a successful access to the GPIO programming model. If the GPIO module's GACR register organized as 32-bit words, the attribute controls for the 4 data bytes in the GACR follow a standard little endian data convention.

## Parameters

<i>base</i>	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
<i>attribute</i>	GPIO checker attribute

## 14.6 FGPIO Driver

### 14.6.1 Overview

This section describes the programming interface of the FGPIO driver. The FGPIO driver configures the FGPIO module and provides a functional interface to build the GPIO application.

Note

FGPIO (Fast GPIO) is only available in a few MCUs. FGPIO and GPIO share the same peripheral but use different registers. FGPIO is closer to the core than the regular GPIO and it's faster to read and write.

### 14.6.2 Typical use case

#### 14.6.2.1 Output Operation

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

#### 14.6.2.2 Input Operation

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

## FGPIO Configuration

- void [FGPIO\\_PinInit](#) (FGPIO\_Type \*base, uint32\_t pin, const [gpio\\_pin\\_config\\_t](#) \*config)  
*Initializes a FGPIO pin used by the board.*

## FGPIO Output Operations

- static void [FGPIO\\_PinWrite](#) (FGPIO\_Type \*base, uint32\_t pin, uint8\_t output)  
*Sets the output level of the multiple FGPIO pins to the logic 1 or 0.*
- static void [FGPIO\\_PortSet](#) (FGPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple FGPIO pins to the logic 1.*
- static void [FGPIO\\_PortClear](#) (FGPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple FGPIO pins to the logic 0.*
- static void [FGPIO\\_PortToggle](#) (FGPIO\_Type \*base, uint32\_t mask)  
*Reverses the current output logic of the multiple FGPIO pins.*

## FGPIO Input Operations

- static uint32\_t [FGPIO\\_PinRead](#) (FGPIO\_Type \*base, uint32\_t pin)  
*Reads the current input value of the FGPIO port.*

## FGPIO Interrupt

- `uint32_t FGPIO_PortGetInterruptFlags (FGPIO_Type *base)`  
*Reads the FGPIO port interrupt status flag.*
- `void FGPIO_PortClearInterruptFlags (FGPIO_Type *base, uint32_t mask)`  
*Clears the multiple FGPIO pin interrupt status flag.*
- `void FGPIO_CheckAttributeBytes (FGPIO_Type *base, gpio_checker_attribute_t attribute)`  
*The FGPIO module supports a device-specific number of data ports, organized as 32-bit words.*

### 14.6.3 Function Documentation

#### 14.6.3.1 void FGPIO\_PinInit ( `FGPIO_Type * base, uint32_t pin, const gpio_pin_config_t * config` )

To initialize the FGPIO driver, define a pin configuration, as either input or output, in the user file. Then, call the `FGPIO_PinInit()` function.

This is an example to define an input pin or an output pin configuration:

```
* Define a digital input pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalInput,
*   0,
* }
* Define a digital output pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalOutput,
*   0,
* }
```

Parameters

<code>base</code>	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
<code>pin</code>	FGPIO port pin number
<code>config</code>	FGPIO pin configuration pointer

#### 14.6.3.2 static void FGPIO\_PinWrite ( `FGPIO_Type * base, uint32_t pin, uint8_t output` ) `[inline], [static]`

Parameters

<i>base</i>	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
<i>pin</i>	GPIO pin number
<i>output</i>	FGPIOpin output logic level. • 0: corresponding pin output low-logic level. • 1: corresponding pin output high-logic level.

#### 14.6.3.3 static void GPIO\_PortSet ( GPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
<i>mask</i>	GPIO pin number macro

#### 14.6.3.4 static void GPIO\_PortClear ( GPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
<i>mask</i>	GPIO pin number macro

#### 14.6.3.5 static void GPIO\_PortToggle ( GPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
<i>mask</i>	GPIO pin number macro

#### 14.6.3.6 static uint32\_t GPIO\_PinRead ( GPIO\_Type \* *base*, uint32\_t *pin* ) [inline], [static]

Parameters

<i>base</i>	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
<i>pin</i>	GPIO pin number

Return values

<i>GPIO</i>	port input value <ul style="list-style-type: none"> <li>• 0: corresponding pin input low-logic level.</li> <li>• 1: corresponding pin input high-logic level.</li> </ul>
-------------	--

#### 14.6.3.7 `uint32_t GPIO_PortGetInterruptFlags ( GPIO_Type * base )`

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level-sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

<i>base</i>	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
-------------	--

Return values

<i>The</i>	current GPIO port interrupt status flags, for example, 0x00010001 means the pin 0 and 17 have the interrupt.
------------	--

#### 14.6.3.8 `void GPIO_PortClearInterruptFlags ( GPIO_Type * base, uint32_t mask )`

Parameters

<i>base</i>	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
<i>mask</i>	GPIO pin number macro

#### 14.6.3.9 `void GPIO_CheckAttributeBytes ( GPIO_Type * base, gpio_checker_attribute_t attribute )`

Each 32-bit data port includes a GACR register, which defines the byte-level attributes required for a successful access to the GPIO programming model. The attribute controls for the 4 data bytes in the GACR follow a standard little endian data convention.

## Parameters

<i>base</i>	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
<i>attribute</i>	GPIO checker attribute

# Chapter 15

## I2C: Inter-Integrated Circuit Driver

### 15.1 Overview

#### Modules

- I2C CMSIS Driver
- I2C DMA Driver
- I2C Driver
- I2C FreeRTOS Driver

## 15.2 I2C Driver

### 15.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Integrated Circuit (I2C) module of MCUXpresso SDK devices.

The I2C driver includes functional APIs and transactional APIs.

Functional APIs target the low-level APIs. Functional APIs can be used for the I2C master/slave initialization/configuration/operation for optimization/customization purpose. Using the functional APIs requires knowing the I2C master peripheral and how to organize functional APIs to meet the application requirements. The I2C functional operation groups provide the functional APIs set.

Transactional APIs target the high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code using the functional APIs or accessing the hardware registers.

Transactional APIs support asynchronous transfer. This means that the functions [I2C\\_MasterTransferNonBlocking\(\)](#) set up the interrupt non-blocking transfer. When the transfer completes, the upper layer is notified through a callback function with the status.

### 15.2.2 Typical use case

#### 15.2.2.1 Master Operation in functional method

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/i2c

#### 15.2.2.2 Master Operation in interrupt transactional method

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/i2c

#### 15.2.2.3 Master Operation in DMA transactional method

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/i2c

#### 15.2.2.4 Slave Operation in functional method

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/i2c

#### 15.2.2.5 Slave Operation in interrupt transactional method

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/i2c

## Data Structures

- struct `i2c_master_config_t`  
*I2C master user configuration.* [More...](#)
- struct `i2c_slave_config_t`  
*I2C slave user configuration.* [More...](#)
- struct `i2c_master_transfer_t`  
*I2C master transfer structure.* [More...](#)
- struct `i2c_master_handle_t`  
*I2C master handle structure.* [More...](#)
- struct `i2c_slave_transfer_t`  
*I2C slave transfer structure.* [More...](#)
- struct `i2c_slave_handle_t`  
*I2C slave handle structure.* [More...](#)

## Macros

- #define `I2C_RETRY_TIMES` 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/  
*Retry times for waiting flag.*
- #define `I2C_MASTER_FACK_CONTROL` 0U /\* Default defines to zero means master will send ack automatically. \*/  
*Mater Fast ack control, control if master needs to manually write ack, this is used to low the speed of transfer for SoCs with feature FSL\_FEATURE\_I2C\_HAS\_DOUBLE\_BUFFERING.*

## Typedefs

- typedef void(\* `i2c_master_transfer_callback_t` )(I2C\_Type \*base, `i2c_master_handle_t` \*handle, `status_t` status, void \*userData)  
*I2C master transfer callback typedef.*
- typedef void(\* `i2c_slave_transfer_callback_t` )(I2C\_Type \*base, `i2c_slave_transfer_t` \*xfer, void \*userData)  
*I2C slave transfer callback typedef.*

## Enumerations

- enum {
 `kStatus_I2C_Busy` = MAKE\_STATUS(kStatusGroup\_I2C, 0),
 `kStatus_I2C_Idle` = MAKE\_STATUS(kStatusGroup\_I2C, 1),
 `kStatus_I2C_Nak` = MAKE\_STATUS(kStatusGroup\_I2C, 2),
 `kStatus_I2C_ArbitrationLost` = MAKE\_STATUS(kStatusGroup\_I2C, 3),
 `kStatus_I2C_Timeout` = MAKE\_STATUS(kStatusGroup\_I2C, 4),
 `kStatus_I2C_Addr_Nak` = MAKE\_STATUS(kStatusGroup\_I2C, 5) }
   
*I2C status return codes.*

- enum `_i2c_flags` {
   
  `kI2C_ReceiveNakFlag` = I2C\_S\_RXAK\_MASK,
   
  `kI2C_IntPendingFlag` = I2C\_S\_IICIF\_MASK,
   
  `kI2C_TransferDirectionFlag` = I2C\_S\_SRW\_MASK,
   
  `kI2C_RangeAddressMatchFlag` = I2C\_S\_RAM\_MASK,
   
  `kI2C_ArbitrationLostFlag` = I2C\_S\_ARBL\_MASK,
   
  `kI2C_BusBusyFlag` = I2C\_S\_BUSY\_MASK,
   
  `kI2C_AddressMatchFlag` = I2C\_S\_IAAS\_MASK,
   
  `kI2C_TransferCompleteFlag` = I2C\_S\_TCF\_MASK,
   
  `kI2C_StopDetectFlag` = I2C\_FLT\_STOPF\_MASK << 8,
   
  `kI2C_StartDetectFlag` = I2C\_FLT\_STARTF\_MASK << 8 }
   
    *I2C peripheral flags.*
- enum `_i2c_interrupt_enable` {
   
  `kI2C_GlobalInterruptEnable` = I2C\_C1\_IICIE\_MASK,
   
  `kI2C_StartStopDetectInterruptEnable` = I2C\_FLT\_SSIE\_MASK }
- enum `i2c_direction_t` {
   
  `kI2C_Write` = 0x0U,
   
  `kI2C_Read` = 0x1U }
   
    *The direction of master and slave transfers.*
- enum `i2c_slave_address_mode_t` {
   
  `kI2C_Address7bit` = 0x0U,
   
  `kI2C_RangeMatch` = 0X2U }
   
    *Addressing mode.*
- enum `_i2c_master_transfer_flags` {
   
  `kI2C_TransferDefaultFlag` = 0x0U,
   
  `kI2C_TransferNoStartFlag` = 0x1U,
   
  `kI2C_TransferRepeatedStartFlag` = 0x2U,
   
  `kI2C_TransferNoStopFlag` = 0x4U }
   
    *I2C transfer control flag.*
- enum `i2c_slave_transfer_event_t` {
   
  `kI2C_SlaveAddressMatchEvent` = 0x01U,
   
  `kI2C_SlaveTransmitEvent` = 0x02U,
   
  `kI2C_SlaveReceiveEvent` = 0x04U,
   
  `kI2C_SlaveTransmitAckEvent` = 0x08U,
   
  `kI2C_SlaveStartEvent` = 0x10U,
   
  `kI2C_SlaveCompletionEvent` = 0x20U,
   
  `kI2C_SlaveGenaralcallEvent` = 0x40U,
   
  `kI2C_SlaveAllEvents` }
   
    *Set of events sent to the callback for nonblocking slave transfers.*
- enum { `kClearFlags` = `kI2C_ArbitrationLostFlag` | `kI2C_IntPendingFlag` | `kI2C_StartDetectFlag` | `kI2C_StopDetectFlag` }
   
    *Common sets of flags used by the driver.*

## Driver version

- #define `FSL_I2C_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 9)`)  
*I2C driver version.*

## Initialization and deinitialization

- void `I2C_MasterInit` (I2C\_Type \*base, const `i2c_master_config_t` \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes the I2C peripheral.*
- void `I2C_SlaveInit` (I2C\_Type \*base, const `i2c_slave_config_t` \*slaveConfig, uint32\_t srcClock\_Hz)  
*Initializes the I2C peripheral.*
- void `I2C_MasterDeinit` (I2C\_Type \*base)  
*De-initializes the I2C master peripheral.*
- void `I2C_SlaveDeinit` (I2C\_Type \*base)  
*De-initializes the I2C slave peripheral.*
- uint32\_t `I2CGetInstance` (I2C\_Type \*base)  
*Get instance number for I2C module.*
- void `I2C_MasterGetDefaultConfig` (`i2c_master_config_t` \*masterConfig)  
*Sets the I2C master configuration structure to default values.*
- void `I2C_SlaveGetDefaultConfig` (`i2c_slave_config_t` \*slaveConfig)  
*Sets the I2C slave configuration structure to default values.*
- static void `I2C_Enable` (I2C\_Type \*base, bool enable)  
*Enables or disables the I2C peripheral operation.*

## Status

- uint32\_t `I2C_MasterGetStatusFlags` (I2C\_Type \*base)  
*Gets the I2C status flags.*
- static uint32\_t `I2C_SlaveGetStatusFlags` (I2C\_Type \*base)  
*Gets the I2C status flags.*
- static void `I2C_MasterClearStatusFlags` (I2C\_Type \*base, uint32\_t statusMask)  
*Clears the I2C status flag state.*
- static void `I2C_SlaveClearStatusFlags` (I2C\_Type \*base, uint32\_t statusMask)  
*Clears the I2C status flag state.*

## Interrupts

- void `I2C_EnableInterrupts` (I2C\_Type \*base, uint32\_t mask)  
*Enables I2C interrupt requests.*
- void `I2C_DisableInterrupts` (I2C\_Type \*base, uint32\_t mask)  
*Disables I2C interrupt requests.*

## DMA Control

- static void [I2C\\_EnableDMA](#) (I2C\_Type \*base, bool enable)  
*Enables/disables the I2C DMA interrupt.*
- static uint32\_t [I2C\\_GetDataRegAddr](#) (I2C\_Type \*base)  
*Gets the I2C tx/rx data register address.*

## Bus Operations

- void [I2C\\_MasterSetBaudRate](#) (I2C\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the I2C master transfer baud rate.*
- [status\\_t I2C\\_MasterStart](#) (I2C\_Type \*base, uint8\_t address, [i2c\\_direction\\_t](#) direction)  
*Sends a START on the I2C bus.*
- [status\\_t I2C\\_MasterStop](#) (I2C\_Type \*base)  
*Sends a STOP signal on the I2C bus.*
- [status\\_t I2C\\_MasterRepeatedStart](#) (I2C\_Type \*base, uint8\_t address, [i2c\\_direction\\_t](#) direction)  
*Sends a REPEATED START on the I2C bus.*
- [status\\_t I2C\\_MasterWriteBlocking](#) (I2C\_Type \*base, const uint8\_t \*txBuff, size\_t txSize, uint32\_t flags)  
*Performs a polling send transaction on the I2C bus.*
- [status\\_t I2C\\_MasterReadBlocking](#) (I2C\_Type \*base, uint8\_t \*rxBuff, size\_t rxSize, uint32\_t flags)  
*Performs a polling receive transaction on the I2C bus.*
- [status\\_t I2C\\_SlaveWriteBlocking](#) (I2C\_Type \*base, const uint8\_t \*txBuff, size\_t txSize)  
*Performs a polling send transaction on the I2C bus.*
- [status\\_t I2C\\_SlaveReadBlocking](#) (I2C\_Type \*base, uint8\_t \*rxBuff, size\_t rxSize)  
*Performs a polling receive transaction on the I2C bus.*
- [status\\_t I2C\\_MasterTransferBlocking](#) (I2C\_Type \*base, [i2c\\_master\\_transfer\\_t](#) \*xfer)  
*Performs a master polling transfer on the I2C bus.*

## Transactional

- void [I2C\\_MasterTransferCreateHandle](#) (I2C\_Type \*base, [i2c\\_master\\_handle\\_t](#) \*handle, [i2c\\_master\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the I2C handle which is used in transactional functions.*
- [status\\_t I2C\\_MasterTransferNonBlocking](#) (I2C\_Type \*base, [i2c\\_master\\_handle\\_t](#) \*handle, [i2c\\_master\\_transfer\\_t](#) \*xfer)  
*Performs a master interrupt non-blocking transfer on the I2C bus.*
- [status\\_t I2C\\_MasterTransferGetCount](#) (I2C\_Type \*base, [i2c\\_master\\_handle\\_t](#) \*handle, size\_t \*count)  
*Gets the master transfer status during a interrupt non-blocking transfer.*
- [status\\_t I2C\\_MasterTransferAbort](#) (I2C\_Type \*base, [i2c\\_master\\_handle\\_t](#) \*handle)  
*Aborts an interrupt non-blocking transfer early.*
- void [I2C\\_MasterTransferHandleIRQ](#) (I2C\_Type \*base, void \*i2cHandle)  
*Master interrupt handler.*
- void [I2C\\_SlaveTransferCreateHandle](#) (I2C\_Type \*base, [i2c\\_slave\\_handle\\_t](#) \*handle, [i2c\\_slave\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the I2C handle which is used in transactional functions.*

- **status\_t I2C\_SlaveTransferNonBlocking** (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle, uint32\_t eventMask)  
*Starts accepting slave transfers.*
- **void I2C\_SlaveTransferAbort** (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle)  
*Aborts the slave transfer.*
- **status\_t I2C\_SlaveTransferGetCount** (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle, size\_t \*count)  
*Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.*
- **void I2C\_SlaveTransferHandleIRQ** (I2C\_Type \*base, void \*i2cHandle)  
*Slave interrupt handler.*

## 15.2.3 Data Structure Documentation

### 15.2.3.1 struct i2c\_master\_config\_t

#### Data Fields

- **bool enableMaster**  
*Enables the I2C peripheral at initialization time.*
- **bool enableStopHold**  
*Controls the stop hold enable.*
- **uint32\_t baudRate\_Bps**  
*Baud rate configuration of I2C peripheral.*
- **uint8\_t glitchFilterWidth**  
*Controls the width of the glitch.*

#### Field Documentation

- (1) **bool i2c\_master\_config\_t::enableMaster**
- (2) **bool i2c\_master\_config\_t::enableStopHold**
- (3) **uint32\_t i2c\_master\_config\_t::baudRate\_Bps**
- (4) **uint8\_t i2c\_master\_config\_t::glitchFilterWidth**

### 15.2.3.2 struct i2c\_slave\_config\_t

#### Data Fields

- **bool enableSlave**  
*Enables the I2C peripheral at initialization time.*
- **bool enableGeneralCall**  
*Enables the general call addressing mode.*
- **bool enableWakeUp**  
*Enables/disables waking up MCU from low-power mode.*
- **bool enableBaudRateCtl**  
*Enables/disables independent slave baud rate on SCL in very fast I2C modes.*
- **uint16\_t slaveAddress**  
*A slave address configuration.*

- **uint16\_t upperAddress**  
A maximum boundary slave address used in a range matching mode.
- **i2c\_slave\_address\_mode\_t addressingMode**  
An addressing mode configuration of i2c\_slave\_address\_mode\_config\_t.
- **uint32\_t sclStopHoldTime\_ns**  
the delay from the rising edge of SCL (I2C clock) to the rising edge of SDA (I2C data) while SCL is high (stop condition), SDA hold time and SCL start hold time are also configured according to the SCL stop hold time.

### Field Documentation

- (1) **bool i2c\_slave\_config\_t::enableSlave**
- (2) **bool i2c\_slave\_config\_t::enableGeneralCall**
- (3) **bool i2c\_slave\_config\_t::enableWakeUp**
- (4) **bool i2c\_slave\_config\_t::enableBaudRateCtl**
- (5) **uint16\_t i2c\_slave\_config\_t::slaveAddress**
- (6) **uint16\_t i2c\_slave\_config\_t::upperAddress**
- (7) **i2c\_slave\_address\_mode\_t i2c\_slave\_config\_t::addressingMode**
- (8) **uint32\_t i2c\_slave\_config\_t::sclStopHoldTime\_ns**

### 15.2.3.3 struct i2c\_master\_transfer\_t

#### Data Fields

- **uint32\_t flags**  
A transfer flag which controls the transfer.
- **uint8\_t slaveAddress**  
7-bit slave address.
- **i2c\_direction\_t direction**  
A transfer direction, read or write.
- **uint32\_t subaddress**  
A sub address.
- **uint8\_t subaddressSize**  
A size of the command buffer.
- **uint8\_t \*volatile data**  
A transfer buffer.
- **volatile size\_t dataSize**  
A transfer size.

### Field Documentation

- (1) **uint32\_t i2c\_master\_transfer\_t::flags**

- (2) `uint8_t i2c_master_transfer_t::slaveAddress`
- (3) `i2c_direction_t i2c_master_transfer_t::direction`
- (4) `uint32_t i2c_master_transfer_t::subaddress`

Transferred MSB first.

- (5) `uint8_t i2c_master_transfer_t::subaddressSize`
- (6) `uint8_t* volatile i2c_master_transfer_t::data`
- (7) `volatile size_t i2c_master_transfer_t::dataSize`

#### 15.2.3.4 struct \_i2c\_master\_handle

I2C master handle typedef.

#### Data Fields

- `i2c_master_transfer_t transfer`  
*I2C master transfer copy.*
- `size_t transferSize`  
*Total bytes to be transferred.*
- `uint8_t state`  
*A transfer state maintained during transfer.*
- `i2c_master_transfer_callback_t completionCallback`  
*A callback function called when the transfer is finished.*
- `void *userData`  
*A callback parameter passed to the callback function.*

#### Field Documentation

- (1) `i2c_master_transfer_t i2c_master_handle_t::transfer`
- (2) `size_t i2c_master_handle_t::transferSize`
- (3) `uint8_t i2c_master_handle_t::state`
- (4) `i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback`
- (5) `void* i2c_master_handle_t::userData`

#### 15.2.3.5 struct i2c\_slave\_transfer\_t

#### Data Fields

- `i2c_slave_transfer_event_t event`  
*A reason that the callback is invoked.*
- `uint8_t *volatile data`

- **volatile size\_t dataSize**  
A transfer buffer.
- **status\_t completionStatus**  
A transfer size.
- **size\_t transferredCount**  
Success or error code describing how the transfer completed.
- **size\_t transferredCount**  
A number of bytes actually transferred since the start or since the last repeated start.

### Field Documentation

- (1) **i2c\_slave\_transfer\_event\_t i2c\_slave\_transfer\_t::event**
  - (2) **uint8\_t\* volatile i2c\_slave\_transfer\_t::data**
  - (3) **volatile size\_t i2c\_slave\_transfer\_t::dataSize**
  - (4) **status\_t i2c\_slave\_transfer\_t::completionStatus**
  - (5) **size\_t i2c\_slave\_transfer\_t::transferredCount**
- Only applies for [kI2C\\_SlaveCompletionEvent](#).

- (5) **size\_t i2c\_slave\_transfer\_t::transferredCount**

### 15.2.3.6 struct \_i2c\_slave\_handle

I2C slave handle typedef.

### Data Fields

- **volatile bool isBusy**  
Indicates whether a transfer is busy.
- **i2c\_slave\_transfer\_t transfer**  
I2C slave transfer copy.
- **uint32\_t eventMask**  
A mask of enabled events.
- **i2c\_slave\_transfer\_callback\_t callback**  
A callback function called at the transfer event.
- **void \* userData**  
A callback parameter passed to the callback.

### Field Documentation

- (1) **volatile bool i2c\_slave\_handle\_t::isBusy**
- (2) **i2c\_slave\_transfer\_t i2c\_slave\_handle\_t::transfer**
- (3) **uint32\_t i2c\_slave\_handle\_t::eventMask**
- (4) **i2c\_slave\_transfer\_callback\_t i2c\_slave\_handle\_t::callback**
- (5) **void\* i2c\_slave\_handle\_t::userData**

## 15.2.4 Macro Definition Documentation

**15.2.4.1 #define FSL\_I2C\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 9))**

**15.2.4.2 #define I2C\_RETRY\_TIMES 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/**

## 15.2.5 Typedef Documentation

**15.2.5.1 typedef void(\* i2c\_master\_transfer\_callback\_t)(I2C\_Type \*base, i2c\_master\_handle\_t \*handle, status\_t status, void \*userData)**

**15.2.5.2 typedef void(\* i2c\_slave\_transfer\_callback\_t)(I2C\_Type \*base, i2c\_slave\_transfer\_t \*xfer, void \*userData)**

## 15.2.6 Enumeration Type Documentation

### 15.2.6.1 anonymous enum

Enumerator

*kStatus\_I2C\_Busy* I2C is busy with current transfer.

*kStatus\_I2C\_Idle* Bus is Idle.

*kStatus\_I2C\_Nak* NAK received during transfer.

*kStatus\_I2C\_ArbitrationLost* Arbitration lost during transfer.

*kStatus\_I2C\_Timeout* Timeout polling status flags.

*kStatus\_I2C\_Addr\_Nak* NAK received during the address probe.

### 15.2.6.2 enum \_i2c\_flags

Note

These enumerations are meant to be OR'd together to form a bit mask.

Enumerator

*kI2C\_ReceiveNakFlag* I2C receive NAK flag.

*kI2C\_IntPendingFlag* I2C interrupt pending flag. This flag can be cleared.

*kI2C\_TransferDirectionFlag* I2C transfer direction flag.

*kI2C\_RangeAddressMatchFlag* I2C range address match flag.

*kI2C\_ArbitrationLostFlag* I2C arbitration lost flag. This flag can be cleared.

*kI2C\_BusBusyFlag* I2C bus busy flag.

*kI2C\_AddressMatchFlag* I2C address match flag.

*kI2C\_TransferCompleteFlag* I2C transfer complete flag.

***kI2C\_StopDetectFlag*** I2C stop detect flag. This flag can be cleared.  
***kI2C\_StartDetectFlag*** I2C start detect flag. This flag can be cleared.

### 15.2.6.3 enum \_i2c\_interrupt\_enable

Enumerator

***kI2C\_GlobalInterruptEnable*** I2C global interrupt.  
***kI2C\_StartStopDetectInterruptEnable*** I2C start&stop detect interrupt.

### 15.2.6.4 enum i2c\_direction\_t

Enumerator

***kI2C\_Write*** Master transmits to the slave.  
***kI2C\_Read*** Master receives from the slave.

### 15.2.6.5 enum i2c\_slave\_address\_mode\_t

Enumerator

***kI2C\_Address7bit*** 7-bit addressing mode.  
***kI2C\_RangeMatch*** Range address match addressing mode.

### 15.2.6.6 enum \_i2c\_master\_transfer\_flags

Enumerator

***kI2C\_TransferDefaultFlag*** A transfer starts with a start signal, stops with a stop signal.  
***kI2C\_TransferNoStartFlag*** A transfer starts without a start signal, only support write only or write+read with no start flag, do not support read only with no start flag.  
***kI2C\_TransferRepeatedStartFlag*** A transfer starts with a repeated start signal.  
***kI2C\_TransferNoStopFlag*** A transfer ends without a stop signal.

### 15.2.6.7 enum i2c\_slave\_transfer\_event\_t

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to [I2C\\_SlaveTransferNonBlocking\(\)](#) to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

## Note

These enumerations are meant to be OR'd together to form a bit mask of events.

## Enumerator

***kI2C\_SlaveAddressMatchEvent*** Received the slave address after a start or repeated start.

***kI2C\_SlaveTransmitEvent*** A callback is requested to provide data to transmit (slave-transmitter role).

***kI2C\_SlaveReceiveEvent*** A callback is requested to provide a buffer in which to place received data (slave-receiver role).

***kI2C\_SlaveTransmitAckEvent*** A callback needs to either transmit an ACK or NACK.

***kI2C\_SlaveStartEvent*** A start/repeated start was detected.

***kI2C\_SlaveCompletionEvent*** A stop was detected or finished transfer, completing the transfer.

***kI2C\_SlaveGeneralCallEvent*** Received the general call address after a start or repeated start.

***kI2C\_SlaveAllEvents*** A bit mask of all available events.

**15.2.6.8 anonymous enum**

## Enumerator

***kClearFlags*** All flags which are cleared by the driver upon starting a transfer.

**15.2.7 Function Documentation****15.2.7.1 void I2C\_MasterInit ( *I2C\_Type \* base*, *const i2c\_master\_config\_t \* masterConfig*, *uint32\_t srcClock\_Hz* )**

Call this API to ungate the I2C clock and configure the I2C with master configuration.

## Note

This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can be custom filled or it can be set with default values by using the [I2C\\_MasterGetDefaultConfig\(\)](#). After calling this API, the master is ready to transfer. This is an example.

```
* i2c_master_config_t config = {
* .enableMaster = true,
* .enableStopHold = false,
* .highDrive = false,
* .baudRate_Bps = 100000,
* .glitchFilterWidth = 0
* };
* I2C_MasterInit(I2C0, &config, 12000000U);
*
```

## Parameters

<i>base</i>	I2C base pointer
<i>masterConfig</i>	A pointer to the master configuration structure
<i>srcClock_Hz</i>	I2C peripheral clock frequency in Hz

**15.2.7.2 void I2C\_SlaveInit ( I2C\_Type \* *base*, const i2c\_slave\_config\_t \* *slaveConfig*, uint32\_t *srcClock\_Hz* )**

Call this API to ungate the I2C clock and initialize the I2C with the slave configuration.

## Note

This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can partly be set with default values by [I2C\\_SlaveGetDefaultConfig\(\)](#) or it can be custom filled by the user. This is an example.

```
* i2c_slave_config_t config = {
*   .enableSlave = true,
*   .enableGeneralCall = false,
*   .addressingMode = kI2C_Address7bit,
*   .slaveAddress = 0x1DU,
*   .enableWakeUp = false,
*   .enableHighDrive = false,
*   .enableBaudRateCtl = false,
*   .sclStopHoldTime_ns = 4000
* };
* I2C_SlaveInit(I2C0, &config, 12000000U);
*
```

## Parameters

<i>base</i>	I2C base pointer
<i>slaveConfig</i>	A pointer to the slave configuration structure
<i>srcClock_Hz</i>	I2C peripheral clock frequency in Hz

**15.2.7.3 void I2C\_MasterDeinit ( I2C\_Type \* *base* )**

Call this API to gate the I2C clock. The I2C master module can't work unless the I2C\_MasterInit is called.

Parameters

<i>base</i>	I2C base pointer
-------------	------------------

#### 15.2.7.4 void I2C\_SlaveDeinit ( I2C\_Type \* *base* )

Calling this API gates the I2C clock. The I2C slave module can't work unless the I2C\_SlaveInit is called to enable the clock.

Parameters

<i>base</i>	I2C base pointer
-------------	------------------

#### 15.2.7.5 uint32\_t I2CGetInstance ( I2C\_Type \* *base* )

Parameters

<i>base</i>	I2C peripheral base address.
-------------	------------------------------

#### 15.2.7.6 void I2C\_MasterGetDefaultConfig ( i2c\_master\_config\_t \* *masterConfig* )

The purpose of this API is to get the configuration structure initialized for use in the I2C\_MasterConfigure(). Use the initialized structure unchanged in the I2C\_MasterConfigure() or modify the structure before calling the I2C\_MasterConfigure(). This is an example.

```
* i2c_master_config_t config;
* I2C_MasterGetDefaultConfig(&config);
*
```

Parameters

<i>masterConfig</i>	A pointer to the master configuration structure.
---------------------	--

#### 15.2.7.7 void I2C\_SlaveGetDefaultConfig ( i2c\_slave\_config\_t \* *slaveConfig* )

The purpose of this API is to get the configuration structure initialized for use in the I2C\_SlaveConfigure(). Modify fields of the structure before calling the I2C\_SlaveConfigure(). This is an example.

```
* i2c_slave_config_t config;
* I2C_SlaveGetDefaultConfig(&config);
*
```

Parameters

<i>slaveConfig</i>	A pointer to the slave configuration structure.
--------------------	---

### 15.2.7.8 static void I2C\_Enable ( I2C\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	I2C base pointer
<i>enable</i>	Pass true to enable and false to disable the module.

### 15.2.7.9 uint32\_t I2C\_MasterGetStatusFlags ( I2C\_Type \* *base* )

Parameters

<i>base</i>	I2C base pointer
-------------	------------------

Returns

status flag, use status flag to AND [\\_i2c\\_flags](#) to get the related status.

### 15.2.7.10 static uint32\_t I2C\_SlaveGetStatusFlags ( I2C\_Type \* *base* ) [inline], [static]

Parameters

<i>base</i>	I2C base pointer
-------------	------------------

Returns

status flag, use status flag to AND [\\_i2c\\_flags](#) to get the related status.

### 15.2.7.11 static void I2C\_MasterClearStatusFlags ( I2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The following status register flags can be cleared kI2C\_ArbitrationLostFlag and kI2C\_IntPendingFlag.

Parameters

<i>base</i>	I2C base pointer
<i>statusMask</i>	<p>The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> <li>• kI2C_StartDetectFlag (if available)</li> <li>• kI2C_StopDetectFlag (if available)</li> <li>• kI2C_ArbitrationLostFlag</li> <li>• kI2C_IntPendingFlagFlag</li> </ul>

#### 15.2.7.12 static void I2C\_SlaveClearStatusFlags ( I2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The following status register flags can be cleared kI2C\_ArbitrationLostFlag and kI2C\_IntPendingFlag

Parameters

<i>base</i>	I2C base pointer
<i>statusMask</i>	<p>The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> <li>• kI2C_StartDetectFlag (if available)</li> <li>• kI2C_StopDetectFlag (if available)</li> <li>• kI2C_ArbitrationLostFlag</li> <li>• kI2C_IntPendingFlagFlag</li> </ul>

#### 15.2.7.13 void I2C\_EnableInterrupts ( I2C\_Type \* *base*, uint32\_t *mask* )

Parameters

<i>base</i>	I2C base pointer
<i>mask</i>	<p>interrupt source The parameter can be combination of the following source if defined:</p> <ul style="list-style-type: none"> <li>• kI2C_GlobalInterruptEnable</li> <li>• kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable</li> <li>• kI2C_SdaTimeoutInterruptEnable</li> </ul>

#### 15.2.7.14 void I2C\_DisableInterrupts ( I2C\_Type \* *base*, uint32\_t *mask* )

Parameters

<i>base</i>	I2C base pointer
<i>mask</i>	interrupt source The parameter can be combination of the following source if defined: <ul style="list-style-type: none"><li>• kI2C_GlobalInterruptEnable</li><li>• kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable</li><li>• kI2C_SdaTimeoutInterruptEnable</li></ul>

### 15.2.7.15 static void I2C\_EnableDMA ( I2C\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	I2C base pointer
<i>enable</i>	true to enable, false to disable

### 15.2.7.16 static uint32\_t I2C\_GetDataRegAddr ( I2C\_Type \* *base* ) [inline], [static]

This API is used to provide a transfer address for I2C DMA transfer configuration.

Parameters

<i>base</i>	I2C base pointer
-------------	------------------

Returns

data register address

### 15.2.7.17 void I2C\_MasterSetBaudRate ( I2C\_Type \* *base*, uint32\_t *baudRate\_Bps*, uint32\_t *srcClock\_Hz* )

Parameters

<i>base</i>	I2C base pointer
<i>baudRate_Bps</i>	the baud rate value in bps
<i>srcClock_Hz</i>	Source clock

### 15.2.7.18 status\_t I2C\_MasterStart ( I2C\_Type \* *base*, uint8\_t *address*, i2c\_direction\_t *direction* )

This function is used to initiate a new master mode transfer by sending the START signal. The slave address is sent following the I2C START signal.

Parameters

<i>base</i>	I2C peripheral base pointer
<i>address</i>	7-bit slave device address.
<i>direction</i>	Master transfer directions(transmit/receive).

Return values

<i>kStatus_Success</i>	Successfully send the start signal.
<i>kStatus_I2C_Busy</i>	Current bus is busy.

### 15.2.7.19 status\_t I2C\_MasterStop ( I2C\_Type \* *base* )

Return values

<i>kStatus_Success</i>	Successfully send the stop signal.
<i>kStatus_I2C_Timeout</i>	Send stop signal failed, timeout.

### 15.2.7.20 status\_t I2C\_MasterRepeatedStart ( I2C\_Type \* *base*, uint8\_t *address*, i2c\_direction\_t *direction* )

Parameters

<i>base</i>	I2C peripheral base pointer
<i>address</i>	7-bit slave device address.
<i>direction</i>	Master transfer directions(transmit/receive).

Return values

<i>kStatus_Success</i>	Successfully send the start signal.
<i>kStatus_I2C_Busy</i>	Current bus is busy but not occupied by current I2C master.

### 15.2.7.21 **status\_t I2C\_MasterWriteBlocking ( I2C\_Type \* *base*, const uint8\_t \* *txBuff*, size\_t *txSize*, uint32\_t *flags* )**

Parameters

<i>base</i>	The I2C peripheral base pointer.
<i>txBuff</i>	The pointer to the data to be transferred.
<i>txSize</i>	The length in bytes of the data to be transferred.
<i>flags</i>	Transfer control flag to decide whether need to send a stop, use kI2C_TransferDefaultFlag to issue a stop and kI2C_TransferNoStop to not send a stop.

Return values

<i>kStatus_Success</i>	Successfully complete the data transmission.
<i>kStatus_I2C_Arbitration-Lost</i>	Transfer error, arbitration lost.
<i>kStatus_I2C_Nak</i>	Transfer error, receive NAK during transfer.

### 15.2.7.22 **status\_t I2C\_MasterReadBlocking ( I2C\_Type \* *base*, uint8\_t \* *rxBuff*, size\_t *rxSize*, uint32\_t *flags* )**

Note

The I2C\_MasterReadBlocking function stops the bus before reading the final byte. Without stopping the bus prior for the final read, the bus issues another read, resulting in garbage data being read into the data register.

Parameters

<i>base</i>	I2C peripheral base pointer.
<i>rxBuff</i>	The pointer to the data to store the received data.
<i>rxSize</i>	The length in bytes of the data to be received.
<i>flags</i>	Transfer control flag to decide whether need to send a stop, use kI2C_TransferDefaultFlag to issue a stop and kI2C_TransferNoStop to not send a stop.

Return values

<i>kStatus_Success</i>	Successfully complete the data transmission.
<i>kStatus_I2C_Timeout</i>	Send stop signal failed, timeout.

### 15.2.7.23 **status\_t I2C\_SlaveWriteBlocking ( I2C\_Type \* *base*, const uint8\_t \* *txBuff*, size\_t *txSize* )**

Parameters

<i>base</i>	The I2C peripheral base pointer.
<i>txBuff</i>	The pointer to the data to be transferred.
<i>txSize</i>	The length in bytes of the data to be transferred.

Return values

<i>kStatus_Success</i>	Successfully complete the data transmission.
<i>kStatus_I2C_ArbitrationLost</i>	Transfer error, arbitration lost.
<i>kStatus_I2C_Nak</i>	Transfer error, receive NAK during transfer.

### 15.2.7.24 **status\_t I2C\_SlaveReadBlocking ( I2C\_Type \* *base*, uint8\_t \* *rxBuff*, size\_t *rxSize* )**

Parameters

<i>base</i>	I2C peripheral base pointer.
<i>rxBuff</i>	The pointer to the data to store the received data.
<i>rxSize</i>	The length in bytes of the data to be received.

Return values

<i>kStatus_Success</i>	Successfully complete data receive.
<i>kStatus_I2C_Timeout</i>	Wait status flag timeout.

### 15.2.7.25 **status\_t I2C\_MasterTransferBlocking ( I2C\_Type \* *base*, i2c\_master\_transfer\_t \* *xfer* )**

## Note

The API does not return until the transfer succeeds or fails due to arbitration lost or receiving a NAK.

## Parameters

<i>base</i>	I2C peripheral base address.
<i>xfer</i>	Pointer to the transfer structure.

## Return values

<i>kStatus_Success</i>	Successfully complete the data transmission.
<i>kStatus_I2C_Busy</i>	Previous transmission still not finished.
<i>kStatus_I2C_Timeout</i>	Transfer error, wait signal timeout.
<i>kStatus_I2C_Arbitration-Lost</i>	Transfer error, arbitration lost.
<i>kStatus_I2C_Nak</i>	Transfer error, receive NAK during transfer.

**15.2.7.26 void I2C\_MasterTransferCreateHandle ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle*, i2c\_master\_transfer\_t *callback*, void \* *userData* )**

## Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to i2c_master_handle_t structure to store the transfer state.
<i>callback</i>	pointer to user callback function.
<i>userData</i>	user parameter passed to the callback function.

**15.2.7.27 status\_t I2C\_MasterTransferNonBlocking ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *xfer* )**

## Note

Calling the API returns immediately after transfer initiates. The user needs to call I2C\_MasterGetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not *kStatus\_I2C\_Busy*, the transfer is finished.

Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to i2c_master_handle_t structure which stores the transfer state.
<i>xfer</i>	pointer to i2c_master_transfer_t structure.

Return values

<i>kStatus_Success</i>	Successfully start the data transmission.
<i>kStatus_I2C_Busy</i>	Previous transmission still not finished.
<i>kStatus_I2C_Timeout</i>	Transfer error, wait signal timeout.

#### 15.2.7.28 status\_t I2C\_MasterTransferGetCount ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to i2c_master_handle_t structure which stores the transfer state.
<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

Return values

<i>kStatus_InvalidArgument</i>	<i>count</i> is Invalid.
<i>kStatus_Success</i>	Successfully return the count.

#### 15.2.7.29 status\_t I2C\_MasterTransferAbort ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle* )

Note

This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to i2c_master_handle_t structure which stores the transfer state

Return values

<i>kStatus_I2C_Timeout</i>	Timeout during polling flag.
<i>kStatus_Success</i>	Successfully abort the transfer.

### 15.2.7.30 void I2C\_MasterTransferHandleIRQ ( *I2C\_Type* \* *base*, *void* \* *i2cHandle* )

Parameters

<i>base</i>	I2C base pointer.
<i>i2cHandle</i>	pointer to <i>i2c_master_handle_t</i> structure.

### 15.2.7.31 void I2C\_SlaveTransferCreateHandle ( *I2C\_Type* \* *base*, *i2c\_slave\_handle\_t* \* *handle*, *i2c\_slave\_transfer\_callback\_t* *callback*, *void* \* *userData* )

Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to <i>i2c_slave_handle_t</i> structure to store the transfer state.
<i>callback</i>	pointer to user callback function.
<i>userData</i>	user parameter passed to the callback function.

### 15.2.7.32 *status\_t* I2C\_SlaveTransferNonBlocking ( *I2C\_Type* \* *base*, *i2c\_slave\_handle\_t* \* *handle*, *uint32\_t* *eventMask* )

Call this API after calling the [I2C\\_SlaveInit\(\)](#) and [I2C\\_SlaveTransferCreateHandle\(\)](#) to start processing transactions driven by an I2C master. The slave monitors the I2C bus and passes events to the callback that was passed into the call to [I2C\\_SlaveTransferCreateHandle\(\)](#). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of *i2c\_slave\_transfer\_event\_t* enumerators for the events you wish to receive. The k-I2C\_SlaveTransmitEvent and kLPI2C\_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the [kI2C\\_SlaveAllEvents](#) constant is provided as a convenient way to enable all events.

Parameters

<i>base</i>	The I2C peripheral base address.
<i>handle</i>	Pointer to i2c_slave_handle_t structure which stores the transfer state.
<i>eventMask</i>	Bit mask formed by OR'ing together <a href="#">i2c_slave_transfer_event_t</a> enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and <a href="#">kI2C_SlaveAllEvents</a> to enable all events.

Return values

<i>kStatus_Success</i>	Slave transfers were successfully started.
<i>kStatus_I2C_Busy</i>	Slave transfers have already been started on this handle.

### 15.2.7.33 void I2C\_SlaveTransferAbort ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle* )

Note

This API can be called at any time to stop slave for handling the bus events.

Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to i2c_slave_handle_t structure which stores the transfer state.

### 15.2.7.34 status\_t I2C\_SlaveTransferGetCount ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to i2c_slave_handle_t structure.
<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

Return values

<i>kStatus_InvalidArgument</i>	count is Invalid.
<i>kStatus_Success</i>	Successfully return the count.

15.2.7.35 void I2C\_SlaveTransferHandleIRQ ( I2C\_Type \* *base*, void \* *i2cHandle* )

## Parameters

<i>base</i>	I2C base pointer.
<i>i2cHandle</i>	pointer to i2c_slave_handle_t structure which stores the transfer state

## 15.3 I2C DMA Driver

### 15.3.1 Overview

#### Data Structures

- struct [i2c\\_master\\_dma\\_handle\\_t](#)  
*I2C master DMA transfer structure. [More...](#)*

#### Typedefs

- [typedef void\(\\* i2c\\_master\\_dma\\_transfer\\_callback\\_t\)\(I2C\\_Type \\*base, i2c\\_master\\_dma\\_handle\\_t \\*handle, status\\_t status, void \\*userData\)](#)  
*I2C master DMA transfer callback typedef.*

#### Driver version

- [#define FSL\\_I2C\\_DMA\\_DRIVER\\_VERSION \(MAKE\\_VERSION\(2, 0, 9\)\)](#)  
*I2C DMA driver version.*

## I2C Block DMA Transfer Operation

- [void I2C\\_MasterTransferCreateHandleDMA \(I2C\\_Type \\*base, i2c\\_master\\_dma\\_handle\\_t \\*handle, i2c\\_master\\_dma\\_transfer\\_callback\\_t callback, void \\*userData, dma\\_handle\\_t \\*dmaHandle\)](#)  
*Initializes the I2C handle which is used in transactional functions.*
- [status\\_t I2C\\_MasterTransferDMA \(I2C\\_Type \\*base, i2c\\_master\\_dma\\_handle\\_t \\*handle, i2c\\_master\\_transfer\\_t \\*xfer\)](#)  
*Performs a master DMA non-blocking transfer on the I2C bus.*
- [status\\_t I2C\\_MasterTransferGetCountDMA \(I2C\\_Type \\*base, i2c\\_master\\_dma\\_handle\\_t \\*handle, size\\_t \\*count\)](#)  
*Gets a master transfer status during a DMA non-blocking transfer.*
- [void I2C\\_MasterTransferAbortDMA \(I2C\\_Type \\*base, i2c\\_master\\_dma\\_handle\\_t \\*handle\)](#)  
*Aborts a master DMA non-blocking transfer early.*

### 15.3.2 Data Structure Documentation

#### 15.3.2.1 struct \_i2c\_master\_dma\_handle

Retry times for waiting flag.

I2C master DMA handle typedef.

## Data Fields

- **i2c\_master\_transfer\_t transfer**  
*I2C master transfer struct.*
- **size\_t transferSize**  
*Total bytes to be transferred.*
- **uint8\_t state**  
*I2C master transfer status.*
- **dma\_handle\_t \* dmaHandle**  
*The DMA handler used.*
- **i2c\_master\_dma\_transfer\_callback\_t completionCallback**  
*A callback function called after the DMA transfer finished.*
- **void \* userData**  
*A callback parameter passed to the callback function.*

## Field Documentation

- (1) **i2c\_master\_transfer\_t i2c\_master\_dma\_handle\_t::transfer**
- (2) **size\_t i2c\_master\_dma\_handle\_t::transferSize**
- (3) **uint8\_t i2c\_master\_dma\_handle\_t::state**
- (4) **dma\_handle\_t\* i2c\_master\_dma\_handle\_t::dmaHandle**
- (5) **i2c\_master\_dma\_transfer\_callback\_t i2c\_master\_dma\_handle\_t::completionCallback**
- (6) **void\* i2c\_master\_dma\_handle\_t::userData**

### 15.3.3 Macro Definition Documentation

15.3.3.1 **#define FSL\_I2C\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 9))**

### 15.3.4 Typedef Documentation

15.3.4.1 **typedef void(\* i2c\_master\_dma\_transfer\_callback\_t)(I2C\_Type \*base, i2c\_master\_dma\_handle\_t \*handle, status\_t status, void \*userData)**

### 15.3.5 Function Documentation

15.3.5.1 **void I2C\_MasterTransferCreateHandleDMA ( I2C\_Type \* base, i2c\_master\_dma\_handle\_t \* handle, i2c\_master\_dma\_transfer\_callback\_t callback, void \* userData, dma\_handle\_t \* dmaHandle )**

Parameters

<i>base</i>	I2C peripheral base address
<i>handle</i>	Pointer to the i2c_master_dma_handle_t structure
<i>callback</i>	Pointer to the user callback function
<i>userData</i>	A user parameter passed to the callback function
<i>dmaHandle</i>	DMA handle pointer

### 15.3.5.2 status\_t I2C\_MasterTransferDMA ( I2C\_Type \* *base*, i2c\_master\_dma\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *xfer* )

Parameters

<i>base</i>	I2C peripheral base address
<i>handle</i>	A pointer to the i2c_master_dma_handle_t structure
<i>xfer</i>	A pointer to the transfer structure of the i2c_master_transfer_t

Return values

<i>kStatus_Success</i>	Successfully completes the data transmission.
<i>kStatus_I2C_Busy</i>	A previous transmission is still not finished.
<i>kStatus_I2C_Timeout</i>	A transfer error, waits for the signal timeout.
<i>kStatus_I2C_Arbitration-Lost</i>	A transfer error, arbitration lost.
<i>kStatus_I2C_Nak</i>	A transfer error, receives NAK during transfer.

### 15.3.5.3 status\_t I2C\_MasterTransferGetCountDMA ( I2C\_Type \* *base*, i2c\_master\_dma\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

<i>base</i>	I2C peripheral base address
<i>handle</i>	A pointer to the i2c_master_dma_handle_t structure

<i>count</i>	A number of bytes transferred so far by the non-blocking transaction.
--------------	---

#### 15.3.5.4 void I2C\_MasterTransferAbortDMA ( I2C\_Type \* *base*, i2c\_master\_dma\_handle\_t \* *handle* )

Parameters

<i>base</i>	I2C peripheral base address
<i>handle</i>	A pointer to the i2c_master_dma_handle_t structure.

## 15.4 I2C FreeRTOS Driver

### 15.4.1 Overview

#### Driver version

- #define `FSL_I2C_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 9)`)  
*I2C FreeRTOS driver version 2.0.9.*

#### I2C RTOS Operation

- `status_t I2C_RTOS_Init` (`i2c_rtos_handle_t *handle, I2C_Type *base, const i2c_master_config_t *masterConfig, uint32_t srcClock_Hz`)  
*Initializes I2C.*
- `status_t I2C_RTOS_Deinit` (`i2c_rtos_handle_t *handle`)  
*Deinitializes the I2C.*
- `status_t I2C_RTOS_Transfer` (`i2c_rtos_handle_t *handle, i2c_master_transfer_t *transfer`)  
*Performs the I2C transfer.*

### 15.4.2 Macro Definition Documentation

#### 15.4.2.1 #define `FSL_I2C_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 9)`)

### 15.4.3 Function Documentation

#### 15.4.3.1 `status_t I2C_RTOS_Init ( i2c_rtos_handle_t * handle, I2C_Type * base, const i2c_master_config_t * masterConfig, uint32_t srcClock_Hz )`

This function initializes the I2C module and the related RTOS context.

Parameters

<code>handle</code>	The RTOS I2C handle, the pointer to an allocated space for RTOS context.
<code>base</code>	The pointer base address of the I2C instance to initialize.
<code>masterConfig</code>	The configuration structure to set-up I2C in master mode.
<code>srcClock_Hz</code>	The frequency of an input clock of the I2C module.

Returns

status of the operation.

### 15.4.3.2 status\_t I2C\_RRTOS\_Deinit ( *i2c\_rtos\_handle\_t \* handle* )

This function deinitializes the I2C module and the related RTOS context.

Parameters

<i>handle</i>	The RTOS I2C handle.
---------------	----------------------

#### **15.4.3.3 status\_t I2C\_RTOS\_Transfer ( i2c\_rtos\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *transfer* )**

This function performs the I2C transfer according to the data given in the transfer structure.

Parameters

<i>handle</i>	The RTOS I2C handle.
<i>transfer</i>	A structure specifying the transfer parameters.

Returns

status of the operation.

## 15.5 I2C CMSIS Driver

This section describes the programming interface of the I2C Cortex Microcontroller Software Interface Standard (CMSIS) driver. This driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method see <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

The I2C CMSIS driver includes transactional APIs.

Transactional APIs are transaction target high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code accessing the hardware registers.

### 15.5.1 I2C CMSIS Driver

#### 15.5.1.1 Master Operation in interrupt transactional method

```
void I2C_MasterSignalEvent_t(uint32_t event)
{
    if (event == ARM_I2C_EVENT_TRANSFER_DONE)
    {
        g_MasterCompletionFlag = true;
    }
}
/*Init I2C0*/
Driver_I2C0.Initialize(I2C_MasterSignalEvent_t);

Driver_I2C0.PowerControl(ARM_POWER_FULL);

/*config transmit speed/
Driver_I2C0.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

/*start transmit*/
Driver_I2C0.MasterTransmit(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

/* Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

#### 15.5.1.2 Master Operation in DMA transactional method

```
void I2C_MasterSignalEvent_t(uint32_t event)
{
    /* Transfer done */
    if (event == ARM_I2C_EVENT_TRANSFER_DONE)
    {
        g_MasterCompletionFlag = true;
    }
}

/* Init DMAMUX and DMA/EDMA. */
DMAMUX_Init(EXAMPLE_I2C_DMAMUX_BASEADDR)
```

```

#if defined(FSL_FEATURE_SOC_DMA_COUNT) && FSL_FEATURE_SOC_DMA_COUNT > 0U
    DMA_Init(EXAMPLE_I2C_DMA_BASEADDR);
#endif /* FSL_FEATURE_SOC_DMA_COUNT */

#if defined(FSL_FEATURE_SOC_EDMA_COUNT) && FSL_FEATURE_SOC_EDMA_COUNT > 0U
    edma_config_t edmaConfig;

    EDMA_GetDefaultConfig(&edmaConfig);
    EDMA_Init(EXAMPLE_I2C_DMA_BASEADDR, &edmaConfig);
#endif /* FSL_FEATURE_SOC_EDMA_COUNT */

/*Init I2C0*/
Driver_I2C0.Initialize(I2C_MasterSignalEvent_t);

Driver_I2C0.PowerControl(ARM_POWER_FULL);

/*config transmit speed*/
Driver_I2C0.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

/*start transfer*/
Driver_I2C0.MasterReceive(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

/* Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;

```

### 15.5.1.3 Slave Operation in interrupt transactional method

```

void I2C_SlaveSignalEvent_t(uint32_t event)
{
    /* Transfer done */
    if (event == ARM_I2C_EVENT_TRANSFER_DONE)
    {
        g_SlaveCompletionFlag = true;
    }
}

/*Init I2C1*/
Driver_I2C1.Initialize(I2C_SlaveSignalEvent_t);

Driver_I2C1.PowerControl(ARM_POWER_FULL);

/*config slave addr*/
Driver_I2C1.Control(ARM_I2C_OWN_ADDRESS, I2C_MASTER_SLAVE_ADDR);

/*start transfer*/
Driver_I2C1.SlaveReceive(g_slave_buff, I2C_DATA_LENGTH);

/* Wait for transfer completed. */
while (!g_SlaveCompletionFlag)
{
}
g_SlaveCompletionFlag = false;

```

# Chapter 16

## IRTC: IRTC Driver

### 16.1 Overview

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

### Data Structures

- struct `irtc_datetime_t`  
*Structure is used to hold the date and time. [More...](#)*
- struct `irtc_daylight_time_t`  
*Structure is used to hold the daylight saving time. [More...](#)*
- struct `irtc_tamper_config_t`  
*Structure is used to define the parameters to configure a RTC tamper event. [More...](#)*
- struct `irtc_config_t`  
*RTC config structure. [More...](#)*

### Enumerations

- enum `irtc_filter_clock_source_t` {  
    `kIRTC_32K` = 0x0U,  
    `kIRTC_512` = 0x1U,  
    `kIRTC_128` = 0x2U,  
    `kIRTC_64` = 0x3U,  
    `kIRTC_16` = 0x4U,  
    `kIRTC_8` = 0x5U,  
    `kIRTC_4` = 0x6U,  
    `kIRTC_2` = 0x7U }  
*IRTC filter clock source options.*
- enum `irtc_tamper_pins_t` {  
    `kIRTC_Tamper_0` = 0U,  
    `kIRTC_Tamper_1`,  
    `kIRTC_Tamper_2`,  
    `kIRTC_Tamper_3` }  
*IRTC Tamper pins.*
- enum `irtc_interrupt_enable_t` {

```

kIRTC_TamperInterruptEnable = RTC_IER_TAMPER_IE_MASK,
kIRTC_AlarmInterruptEnable = RTC_IER_ALM_IE_MASK,
kIRTC_DayInterruptEnable = RTC_IER_DAY_IE_MASK,
kIRTC_HourInterruptEnable = RTC_IER_HOUR_IE_MASK,
kIRTC_MinInterruptEnable = RTC_IER_MIN_IE_MASK,
kIRTC_1hzInterruptEnable = RTC_IER_IE_1HZ_MASK,
kIRTC_2hzInterruptEnable = RTC_IER_IE_2HZ_MASK,
kIRTC_4hzInterruptEnable = RTC_IER_IE_4HZ_MASK,
kIRTC_8hzInterruptEnable = RTC_IER_IE_8HZ_MASK,
kIRTC_16hzInterruptEnable = RTC_IER_IE_16HZ_MASK,
kIRTC_32hzInterruptEnable = RTC_IER_IE_32HZ_MASK,
kIRTC_64hzInterruptEnable = RTC_IER_IE_64HZ_MASK,
kIRTC_128hzInterruptEnable = RTC_IER_IE_128HZ_MASK,
kIRTC_256hzInterruptEnable = RTC_IER_IE_256HZ_MASK,
kIRTC_512hzInterruptEnable = RTC_IER_IE_512HZ_MASK }

```

*List of IRTC interrupts.*

- enum `irtc_status_flags_t` {

```

kIRTC_TamperFlag = RTC_ISR_TAMPER_IS_MASK,
kIRTC_AlarmFlag = RTC_ISR_ALM_IS_MASK,
kIRTC_DayFlag = RTC_ISR_DAY_IS_MASK,
kIRTC_HourFlag = RTC_ISR_HOUR_IS_MASK,
kIRTC_MinFlag = RTC_ISR_MIN_IS_MASK,
kIRTC_1hzFlag = RTC_ISR_IS_1HZ_MASK,
kIRTC_2hzFlag = RTC_ISR_IS_2HZ_MASK,
kIRTC_4hzFlag = RTC_ISR_IS_4HZ_MASK,
kIRTC_8hzFlag = RTC_ISR_IS_8HZ_MASK,
kIRTC_16hzFlag = RTC_ISR_IS_16HZ_MASK,
kIRTC_32hzFlag = RTC_ISR_IS_32HZ_MASK,
kIRTC_64hzFlag = RTC_ISR_IS_64HZ_MASK,
kIRTC_128hzFlag = RTC_ISR_IS_128HZ_MASK,
kIRTC_256hzFlag = RTC_ISR_IS_256HZ_MASK,
kIRTC_512hzFlag = RTC_ISR_IS_512HZ_MASK,
kIRTC_InvalidFlag = (RTC_STATUS_INVAL_BIT_MASK << 16U),
kIRTC_WriteProtFlag = (RTC_STATUS_WRITE_PROT_EN_MASK << 16U),
kIRTC_CpuLowVoltFlag = (RTC_STATUS_CPU_LOW_VOLT_MASK << 16U),
kIRTC_ResetSrcFlag = (RTC_STATUS_RST_SRC_MASK << 16U),
kIRTC_CmpIntFlag = (RTC_STATUS_CMP_INT_MASK << 16U),
kIRTC_BusErrFlag = (RTC_STATUS_BUS_ERR_MASK << 16U),
kIRTC_CmpDoneFlag = (RTC_STATUS_CMP_DONE_MASK << 16U) }

```

*List of IRTC flags.*

- enum `irtc_alarm_match_t` {

```

kRTC_MatchSecMinHr = 0U,
kRTC_MatchSecMinHrDay = 1U,
kRTC_MatchSecMinHrDayMnth = 2U,
kRTC_MatchSecMinHrDayMnthYr = 3U } 
```

- *IRTC alarm match options.*
- enum `irtc_osc_cap_load_t` {
   
    kIRTC\_Capacitor2p = (1U << 1U),
   
    kIRTC\_Capacitor4p = (1U << 2U),
   
    kIRTC\_Capacitor8p = (1U << 3U),
   
    kIRTC\_Capacitor16p = (1U << 4U) }
- List of RTC Oscillator capacitor load settings.*
- enum `irtc_clockout_sel_t` {
   
    kIRTC\_ClkoutNo = 0U,
   
    kIRTC\_ClkoutFine1Hz,
   
    kIRTC\_Clkout32kHz,
   
    kIRTC\_ClkoutCoarse1Hz }
- IRTC clockout select.*

## Functions

- static void `IRTC_SetOscCapLoad` (RTC\_Type \*base, uint16\_t capLoad)
   
    *This function sets the specified capacitor configuration for the RTC oscillator.*
- status\_t `IRTC_SetWriteProtection` (RTC\_Type \*base, bool lock)
   
    *Locks or unlocks IRTC registers for write access.*
- static void `IRTC_Reset` (RTC\_Type \*base)
   
    *Performs a software reset on the IRTC module.*
- static void `IRTC_Enable32kClkDuringRegisterWrite` (RTC\_Type \*base, bool enable)
   
    *Enable/disable 32 kHz RTC OSC clock during RTC register write.*
- void `IRTC_ConfigClockOut` (RTC\_Type \*base, `irtc_clockout_sel_t` clkOut)
   
    *Select which clock to output from RTC.*
- static uint8\_t `IRTC_GetTamperStatusFlag` (RTC\_Type \*base)
   
    *Gets the IRTC Tamper status flags.*
- static void `IRTC_ClearTamperStatusFlag` (RTC\_Type \*base)
   
    *Gets the IRTC Tamper status flags.*
- static void `IRTC_SetTamperConfigurationOver` (RTC\_Type \*base)
   
    *Set tamper configuration over.*

## Driver version

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

## Initialization and deinitialization

- status\_t `IRTC_Init` (RTC\_Type \*base, const `irtc_config_t` \*config)
   
    *Ungates the IRTC clock and configures the peripheral for basic operation.*
- static void `IRTC_Deinit` (RTC\_Type \*base)
   
    *Gate the IRTC clock.*
- void `IRTC_GetDefaultConfig` (`irtc_config_t` \*config)
   
    *Fill in the IRTC config struct with the default settings.*

## Current Time & Alarm

- status\_t `IRTC_SetDatetime` (RTC\_Type \*base, const `irtc_datetime_t` \*datetime)

- Sets the IRTC date and time according to the given time structure.
- void **IRTC\_GetDatetime** (RTC\_Type \*base, **irtc\_datetime\_t** \*datetime)  
Gets the IRTC time and stores it in the given time structure.
- **status\_t IRTC\_SetAlarm** (RTC\_Type \*base, const **irtc\_datetime\_t** \*alarmTime)  
Sets the IRTC alarm time.
- void **IRTC\_GetAlarm** (RTC\_Type \*base, **irtc\_datetime\_t** \*datetime)  
Returns the IRTC alarm time.

## Interrupt Interface

- static void **IRTC\_EnableInterrupts** (RTC\_Type \*base, uint32\_t mask)  
Enables the selected IRTC interrupts.
- static void **IRTC\_DisableInterrupts** (RTC\_Type \*base, uint32\_t mask)  
Disables the selected IRTC interrupts.
- static uint32\_t **IRTC\_GetEnabledInterrupts** (RTC\_Type \*base)  
Gets the enabled IRTC interrupts.

## Status Interface

- static uint32\_t **IRTC\_GetStatusFlags** (RTC\_Type \*base)  
Gets the IRTC status flags.
- static void **IRTC\_ClearStatusFlags** (RTC\_Type \*base, uint32\_t mask)  
Clears the IRTC status flags.

## Daylight Savings Interface

- void **IRTC\_SetDaylightTime** (RTC\_Type \*base, const **irtc\_daylight\_time\_t** \*datetime)  
Sets the IRTC daylight savings start and stop date and time.
- void **IRTC\_GetDaylightTime** (RTC\_Type \*base, **irtc\_daylight\_time\_t** \*datetime)  
Gets the IRTC daylight savings time and stores it in the given time structure.

## Time Compensation Interface

- void **IRTC\_SetCoarseCompensation** (RTC\_Type \*base, uint8\_t compensationValue, uint8\_t compensationInterval)  
Enables the coarse compensation and sets the value in the IRTC compensation register.
- void **IRTC\_SetFineCompensation** (RTC\_Type \*base, uint8\_t integralValue, uint8\_t fractionValue, bool accumulateFractional)  
Enables the fine compensation and sets the value in the IRTC compensation register.

## Tamper Interface

- void **IRTC\_SetTamperParams** (RTC\_Type \*base, **irtc\_tamper\_pins\_t** tamperNumber, const **irtc\_tamper\_config\_t** \*tamperConfig)  
This function allows configuring the four tamper inputs.

## 16.2 Data Structure Documentation

### 16.2.1 struct irtc\_datetime\_t

#### Data Fields

- `uint16_t year`  
*Range from 1984 to 2239.*
- `uint8_t month`  
*Range from 1 to 12.*
- `uint8_t day`  
*Range from 1 to 31 (depending on month).*
- `uint8_t weekDay`  
*Range from 0(Sunday) to 6(Saturday).*
- `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 irtc_datetime_t::year`
- (2) `uint8_t irtc_datetime_t::month`
- (3) `uint8_t irtc_datetime_t::day`
- (4) `uint8_t irtc_datetime_t::weekDay`
- (5) `uint8_t irtc_datetime_t::hour`
- (6) `uint8_t irtc_datetime_t::minute`
- (7) `uint8_t irtc_datetime_t::second`

### 16.2.2 struct irtc\_daylight\_time\_t

#### Data Fields

- `uint8_t startMonth`  
*Range from 1 to 12.*
- `uint8_t endMonth`  
*Range from 1 to 12.*
- `uint8_t startDay`  
*Range from 1 to 31 (depending on month)*
- `uint8_t endDay`  
*Range from 1 to 31 (depending on month)*
- `uint8_t startHour`  
*Range from 0 to 23.*

- `uint8_t endHour`  
*Range from 0 to 23.*

### 16.2.3 `struct irtc_tamper_config_t`

#### Data Fields

- `bool pinPolarity`  
*true: tamper has active low polarity; false: active high polarity*
- `irtc_filter_clock_source_t filterClk`  
*Clock source for the tamper filter.*
- `uint8_t filterDuration`  
*Tamper filter duration.*

#### Field Documentation

##### (1) `uint8_t irtc_tamper_config_t::filterDuration`

### 16.2.4 `struct irtc_config_t`

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

The config struct can be made const so it resides in flash

#### Data Fields

- `bool wakeupSelect`  
*true: Tamper pin 0 is used to wakeup the chip; false: Tamper pin 0 is used as the tamper pin*
- `bool timerStdMask`  
*true: Sampling clocks gated in standby mode; false: Sampling clocks not gated*
- `irtc_alarm_match_t alrmMatch`  
*Pick one option from enumeration :: irtc\_alarm\_match\_t.*

### 16.3 Macro Definition Documentation

#### 16.3.1 `#define FSL_IRTC_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

### 16.4 Enumeration Type Documentation

#### 16.4.1 `enum irtc_filter_clock_source_t`

Enumerator

***KIRTC\_32K*** Use 32 kHz clock source for the tamper filter.

***kIRTC\_512*** Use 512 Hz clock source for the tamper filter.  
***kIRTC\_128*** Use 128 Hz clock source for the tamper filter.  
***kIRTC\_64*** Use 64 Hz clock source for the tamper filter.  
***kIRTC\_16*** Use 16 Hz clock source for the tamper filter.  
***kIRTC\_8*** Use 8 Hz clock source for the tamper filter.  
***kIRTC\_4*** Use 4 Hz clock source for the tamper filter.  
***kIRTC\_2*** Use 2 Hz clock source for the tamper filter.

#### 16.4.2 enum irtc\_tamper\_pins\_t

Enumerator

***kIRTC\_Tamper\_0*** External Tamper 0.  
***kIRTC\_Tamper\_1*** External Tamper 1.  
***kIRTC\_Tamper\_2*** External Tamper 2.  
***kIRTC\_Tamper\_3*** Internal tamper, does not have filter configuration.

#### 16.4.3 enum irtc\_interrupt\_enable\_t

Enumerator

***kIRTC\_TamperInterruptEnable*** Tamper Interrupt Enable.  
***kIRTC\_AlarmInterruptEnable*** Alarm Interrupt Enable.  
***kIRTC\_DayInterruptEnable*** Days Interrupt Enable.  
***kIRTC\_HourInterruptEnable*** Hours Interrupt Enable.  
***kIRTC\_MinInterruptEnable*** Minutes Interrupt Enable.  
***kIRTC\_1hzInterruptEnable*** 1 Hz interval Interrupt Enable  
***kIRTC\_2hzInterruptEnable*** 2 Hz interval Interrupt Enable  
***kIRTC\_4hzInterruptEnable*** 4 Hz interval Interrupt Enable  
***kIRTC\_8hzInterruptEnable*** 8 Hz interval Interrupt Enable  
***kIRTC\_16hzInterruptEnable*** 16 Hz interval Interrupt Enable  
***kIRTC\_32hzInterruptEnable*** 32 Hz interval Interrupt Enable  
***kIRTC\_64hzInterruptEnable*** 64 Hz interval Interrupt Enable  
***kIRTC\_128hzInterruptEnable*** 128 Hz interval Interrupt Enable  
***kIRTC\_256hzInterruptEnable*** 256 Hz interval Interrupt Enable  
***kIRTC\_512hzInterruptEnable*** 512 Hz interval Interrupt Enable

#### 16.4.4 enum irtc\_status\_flags\_t

Enumerator

***kIRTC\_TamperFlag*** Tamper Status flag.

*kIRTC\_AlarmFlag* Alarm Status flag.  
*kIRTC\_DayFlag* Days Status flag.  
*kIRTC\_HourFlag* Hour Status flag.  
*kIRTC\_MinFlag* Minutes Status flag.  
*kIRTC\_1hzFlag* 1 Hz interval status flag  
*kIRTC\_2hzFlag* 2 Hz interval status flag  
*kIRTC\_4hzFlag* 4 Hz interval status flag  
*kIRTC\_8hzFlag* 8 Hz interval status flag  
*kIRTC\_16hzFlag* 16 Hz interval status flag  
*kIRTC\_32hzFlag* 32 Hz interval status flag  
*kIRTC\_64hzFlag* 64 Hz interval status flag  
*kIRTC\_128hzFlag* 128 Hz interval status flag  
*kIRTC\_256hzFlag* 256 Hz interval status flag  
*kIRTC\_512hzFlag* 512 Hz interval status flag  
*kIRTC\_InvalidFlag* Indicates if time/date counters are invalid.  
*kIRTC\_WriteProtFlag* Write protect enable status flag.  
*kIRTC\_CpuLowVoltFlag* CPU low voltage warning flag.  
*kIRTC\_ResetSrcFlag* Reset source flag.  
*kIRTC\_CmpIntFlag* Compensation interval status flag.  
*kIRTC\_BusErrFlag* Bus error flag.  
*kIRTC\_CmpDoneFlag* Compensation done flag.

#### 16.4.5 enum irtc\_alarm\_match\_t

Enumerator

*kRTC\_MatchSecMinHr* Only match second, minute and hour.  
*kRTC\_MatchSecMinHrDay* Only match second, minute, hour and day.  
*kRTC\_MatchSecMinHrDayMnth* Only match second, minute, hour, day and month.  
*kRTC\_MatchSecMinHrDayMnthYr* Only match second, minute, hour, day, month and year.

#### 16.4.6 enum irtc\_osc\_cap\_load\_t

Enumerator

*kIRTC\_Capacitor2p* 2pF capacitor load  
*kIRTC\_Capacitor4p* 4pF capacitor load  
*kIRTC\_Capacitor8p* 8pF capacitor load  
*kIRTC\_Capacitor16p* 16pF capacitor load

### 16.4.7 enum irtc\_clockout\_sel\_t

Enumerator

*kIRTC\_ClkoutNo* No clock out.  
*kIRTC\_ClkoutFine1Hz* clock out fine 1Hz  
*kIRTC\_Clkout32kHz* clock out 32.768kHz  
*kIRTC\_ClkoutCoarse1Hz* clock out coarse 1Hz

## 16.5 Function Documentation

### 16.5.1 status\_t IRTC\_Init( RTC\_Type \* *base*, const irtc\_config\_t \* *config* )

This function initiates a soft-reset of the IRTC module, this has not effect on DST, calendaring, standby time and tamper detect registers.

Note

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

Parameters

<i>base</i>	IRTC peripheral base address
<i>config</i>	Pointer to user's IRTC config structure.

Returns

kStatus\_Fail if we cannot disable register write protection

### 16.5.2 static void IRTC\_Deinit( RTC\_Type \* *base* ) [inline], [static]

Parameters

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

### 16.5.3 void IRTC\_GetDefaultConfig( irtc\_config\_t \* *config* )

The default values are:

```
*     config->wakeupSelect = true;
*     config->timerStdMask = false;
*     config->alarmMatch = kRTC_MatchSecMinHr;
*
```

Parameters

<i>config</i>	Pointer to user's IRTC config structure.
---------------	--

#### 16.5.4 status\_t IRTC\_SetDatetime ( RTC\_Type \* *base*, const irtc\_datetime\_t \* *datetime* )

The IRTC counter is started after the time is set.

Parameters

<i>base</i>	IRTC peripheral base address
<i>datetime</i>	Pointer to structure where the date and time details to set are stored

Returns

kStatus\_Success: success in setting the time and starting the IRTC  
kStatus\_InvalidArgument: failure. An error occurs because the datetime format is incorrect.

#### 16.5.5 void IRTC\_GetDatetime ( RTC\_Type \* *base*, irtc\_datetime\_t \* *datetime* )

Parameters

<i>base</i>	IRTC peripheral base address
<i>datetime</i>	Pointer to structure where the date and time details are stored.

#### 16.5.6 status\_t IRTC\_SetAlarm ( RTC\_Type \* *base*, const irtc\_datetime\_t \* *alarmTime* )

Parameters

<i>base</i>	RTC peripheral base address
<i>alarmTime</i>	Pointer to structure where the alarm time is stored.

Note

weekDay field of alarmTime is not used during alarm match and should be set to 0

Returns

kStatus\_Success: success in setting the alarm  
 kStatus\_InvalidArgument: error in setting the alarm.  
 Error occurs because the alarm datetime format is incorrect.

### **16.5.7 void IRTC\_GetAlarm ( RTC\_Type \* *base*, irtc\_datetime\_t \* *datetime* )**

Parameters

<i>base</i>	RTC peripheral base address
<i>datetime</i>	Pointer to structure where the alarm date and time details are stored.

### **16.5.8 static void IRTC\_EnableInterrupts ( RTC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

<i>base</i>	IRTC peripheral base address
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">irtc_interrupt_enable_t</a>

### **16.5.9 static void IRTC\_DisableInterrupts ( RTC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

<i>base</i>	IRTC peripheral base address
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">irtc_interrupt_enable_t</a>

### **16.5.10 static uint32\_t IRTC\_GetEnabledInterrupts ( RTC\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

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

### 16.5.11 static uint32\_t IRTC\_GetStatusFlags ( RTC\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

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

### 16.5.12 static void IRTC\_ClearStatusFlags ( RTC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

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

### 16.5.13 static void IRTC\_SetOscCapLoad ( RTC\_Type \* *base*, uint16\_t *capLoad* ) [inline], [static]

Parameters

<i>base</i>	IRTC peripheral base address
<i>capLoad</i>	Oscillator loads to enable. This is a logical OR of members of the enumeration <a href="#">irtc_osc_cap_load_t</a>

**16.5.14 status\_t IRTC\_SetWriteProtection ( RTC\_Type \* *base*, bool *lock* )**

Note

When the registers are unlocked, they remain in unlocked state for 2 seconds, after which they are locked automatically. After power-on-reset, the registers come out unlocked and they are locked automatically 15 seconds after power on.

Parameters

<i>base</i>	IRTC peripheral base address
<i>lock</i>	true: Lock IRTC registers; false: Unlock IRTC registers.

Returns

kStatus\_Success: if lock or unlock operation is successful  
kStatus\_Fail: if lock or unlock operation fails even after multiple retry attempts

**16.5.15 static void IRTC\_Reset ( RTC\_Type \* *base* ) [inline], [static]**

Clears contents of alarm, interrupt (status and enable except tamper interrupt enable bit) registers, STATUS[CMP\_DONE] and STATUS[BUS\_ERR]. This has no effect on DST, calendaring, standby time and tamper detect registers.

Parameters

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

**16.5.16 static void IRTC\_Enable32kClkDuringRegisterWrite ( RTC\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

<i>base</i>	IRTC peripheral base address
<i>enable</i>	Enable/disable 32 kHz RTC OSC clock. <ul style="list-style-type: none"><li>• true: Enables the oscillator.</li><li>• false: Disables the oscillator.</li></ul>

### 16.5.17 void IRTC\_ConfigClockOut ( RTC\_Type \* *base*, irtc\_clockout\_sel\_t *clkOut* )

Select which clock to output from RTC for other modules to use inside SoC, for example, RTC subsystem needs RTC to output 1HZ clock for sub-second counter.

Parameters

<i>base</i>	IRTC peripheral base address
<i>cloOut</i>	select clock to use for output,

### 16.5.18 static uint8\_t IRTC\_GetTamperStatusFlag ( RTC\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The Tamper status value.

### 16.5.19 static void IRTC\_ClearTamperStatusFlag ( RTC\_Type \* *base* ) [inline], [static]

Parameters

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

**16.5.20 static void IRTC\_SetTamperConfigurationOver ( RTC\_Type \* *base* )  
[inline], [static]**

Note that this API is needed after call IRTC\_SetTamperParams to configure tamper events to notify IRTC module that tamper configuration process is over.

Parameters

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

#### 16.5.21 void IRTC\_SetDaylightTime ( RTC\_Type \* *base*, const irtc\_daylight\_time\_t \* *datetime* )

It also enables the daylight saving bit in the IRTC control register

Parameters

<i>base</i>	IRTC peripheral base address
<i>datetime</i>	Pointer to a structure where the date and time details are stored.

#### 16.5.22 void IRTC\_GetDaylightTime ( RTC\_Type \* *base*, irtc\_daylight\_time\_t \* *datetime* )

Parameters

<i>base</i>	IRTC peripheral base address
<i>datetime</i>	Pointer to a structure where the date and time details are stored.

#### 16.5.23 void IRTC\_SetCoarseCompensation ( RTC\_Type \* *base*, uint8\_t *compensationValue*, uint8\_t *compensationInterval* )

Parameters

<i>base</i>	IRTC peripheral base address
<i>compensationValue</i>	Compensation value is a 2's complement value.
<i>compensationInterval</i>	Compensation interval.

#### 16.5.24 void IRTC\_SetFineCompensation ( RTC\_Type \* *base*, uint8\_t *integralValue*, uint8\_t *fractionValue*, bool *accumulateFractional* )

## Parameters

<i>base</i>	The IRTC peripheral base address
<i>integralValue</i>	Compensation integral value; twos complement value of the integer part
<i>fractionValue</i>	Compensation fraction value expressed as number of clock cycles of a fixed 4.-194304Mhz clock that have to be added.
<i>accumulate-Fractional</i>	Flag indicating if we want to add to previous fractional part; true: Add to previously accumulated fractional part, false: Start afresh and overwrite current value

**16.5.25 void IRTC\_SetTamperParams ( RTC\_Type \* *base*, irtc\_tamper\_pins\_t *tamperNumber*, const irtc\_tamper\_config\_t \* *tamperConfig* )**

The function configures the filter properties for the three external tampers. It also sets up active/passive and direction of the tamper bits, which are not available on all platforms.

## Note

This function programs the tamper filter parameters. The user must gate the 32K clock to the RTC before calling this function. It is assumed that the time and date are set after this and the tamper parameters do not require to be changed again later.

## Parameters

<i>base</i>	The IRTC peripheral base address
<i>tamperNumber</i>	The IRTC tamper input to configure
<i>tamperConfig</i>	The IRTC tamper properties

# Chapter 17

## LLWU: Low-Leakage Wakeup Unit Driver

### 17.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low-Leakage Wakeup Unit (LLWU) module of MCUXpresso SDK devices. The LLWU module allows the user to select external pin sources and internal modules as a wake-up source from low-leakage power modes.

### 17.2 External wakeup pins configurations

Configures the external wakeup pins' working modes, gets, and clears the wake pin flags. External wakeup pins are accessed by the `pinIndex`, which is started from 1. Numbers of the external pins depend on the SoC configuration.

### 17.3 Internal wakeup modules configurations

Enables/disables the internal wakeup modules and gets the module flags. Internal modules are accessed by `moduleIndex`, which is started from 1. Numbers of external pins depend the on SoC configuration.

### 17.4 Digital pin filter for external wakeup pin configurations

Configures the digital pin filter of the external wakeup pins' working modes, gets, and clears the pin filter flags. Digital pin filters are accessed by the `filterIndex`, which is started from 1. Numbers of external pins depend on the SoC configuration.

## Data Structures

- struct `llwu_version_id_t`  
*IP version ID definition. [More...](#)*
- struct `llwu_param_t`  
*IP parameter definition. [More...](#)*
- struct `llwu_external_pin_filter_mode_t`  
*An external input pin filter control structure. [More...](#)*

## Enumerations

- enum `llwu_external_pin_mode_t` {  
  `kLLWU_ExternalPinDisable` = 0U,  
  `kLLWU_ExternalPinRisingEdge` = 1U,  
  `kLLWU_ExternalPinFallingEdge` = 2U,  
  `kLLWU_ExternalPinAnyEdge` = 3U }  
*External input pin control modes.*

- enum llwu\_pin\_filter\_mode\_t {
 kLLWU\_PinFilterDisable = 0U,
 kLLWU\_PinFilterRisingEdge = 1U,
 kLLWU\_PinFilterFallingEdge = 2U,
 kLLWU\_PinFilterAnyEdge = 3U }

*Digital filter control modes.*

## Driver version

- #define FSL\_LLWU\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 5))  
*LLWU driver version.*

## Low-Leakage Wakeup Unit Control APIs

- static void LLWU\_GetVersionId (LLWU\_Type \*base, llwu\_version\_id\_t \*versionId)  
*Gets the LLWU version ID.*
- static void LLWU\_GetParam (LLWU\_Type \*base, llwu\_param\_t \*param)  
*Gets the LLWU parameter.*
- void LLWU\_SetExternalWakeUpPinMode (LLWU\_Type \*base, uint32\_t pinIndex, llwu\_external\_pin\_mode\_t pinMode)  
*Sets the external input pin source mode.*
- bool LLWU\_GetExternalWakeUpPinFlag (LLWU\_Type \*base, uint32\_t pinIndex)  
*Gets the external wakeup source flag.*
- void LLWU\_ClearExternalWakeUpPinFlag (LLWU\_Type \*base, uint32\_t pinIndex)  
*Clears the external wakeup source flag.*
- static void LLWU\_EnableInternalModuleInterruptWakeup (LLWU\_Type \*base, uint32\_t moduleIndex, bool enable)  
*Enables/disables the internal module source.*
- static bool LLWU\_GetInternalWakeUpModuleFlag (LLWU\_Type \*base, uint32\_t moduleIndex)  
*Gets the external wakeup source flag.*
- static void LLWU\_EnableInternalModuleDmaRequestWakeup (LLWU\_Type \*base, uint32\_t moduleIndex, bool enable)  
*Enables/disables the internal module DMA wakeup source.*
- void LLWU\_SetPinFilterMode (LLWU\_Type \*base, uint32\_t filterIndex, llwu\_external\_pin\_filter\_mode\_t filterMode)  
*Sets the pin filter configuration.*
- bool LLWU\_GetPinFilterFlag (LLWU\_Type \*base, uint32\_t filterIndex)  
*Gets the pin filter configuration.*
- void LLWU\_ClearPinFilterFlag (LLWU\_Type \*base, uint32\_t filterIndex)  
*Clears the pin filter configuration.*
- void LLWU\_SetResetPinMode (LLWU\_Type \*base, bool pinEnable, bool pinFilterEnable)  
*Sets the reset pin mode.*
- #define INTERNAL\_WAKEUP\_MODULE\_FLAG\_REG MF5

## 17.5 Data Structure Documentation

### 17.5.1 struct llwu\_version\_id\_t

#### Data Fields

- `uint16_t feature`  
*A feature specification number.*
- `uint8_t minor`  
*The minor version number.*
- `uint8_t major`  
*The major version number.*

#### Field Documentation

- (1) `uint16_t llwu_version_id_t::feature`
- (2) `uint8_t llwu_version_id_t::minor`
- (3) `uint8_t llwu_version_id_t::major`

### 17.5.2 struct llwu\_param\_t

#### Data Fields

- `uint8_t filters`  
*A number of the pin filter.*
- `uint8_t dmas`  
*A number of the wakeup DMA.*
- `uint8_t modules`  
*A number of the wakeup module.*
- `uint8_t pins`  
*A number of the wake up pin.*

#### Field Documentation

- (1) `uint8_t llwu_param_t::filters`
- (2) `uint8_t llwu_param_t::dmas`
- (3) `uint8_t llwu_param_t::modules`
- (4) `uint8_t llwu_param_t::pins`

### 17.5.3 struct llwu\_external\_pin\_filter\_mode\_t

#### Data Fields

- `uint32_t pinIndex`  
*A pin number.*

- `llwu_pin_filter_mode_t filterMode`  
*Filter mode.*

## 17.6 Macro Definition Documentation

### 17.6.1 `#define FSL_LLWU_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))`

## 17.7 Enumeration Type Documentation

### 17.7.1 `enum llwu_external_pin_mode_t`

Enumerator

`kLLWU_ExternalPinDisable` Pin disabled as a wakeup input.

`kLLWU_ExternalPinRisingEdge` Pin enabled with the rising edge detection.

`kLLWU_ExternalPinFallingEdge` Pin enabled with the falling edge detection.

`kLLWU_ExternalPinAnyEdge` Pin enabled with any change detection.

### 17.7.2 `enum llwu_pin_filter_mode_t`

Enumerator

`kLLWU_PinFilterDisable` Filter disabled.

`kLLWU_PinFilterRisingEdge` Filter positive edge detection.

`kLLWU_PinFilterFallingEdge` Filter negative edge detection.

`kLLWU_PinFilterAnyEdge` Filter any edge detection.

## 17.8 Function Documentation

### 17.8.1 `static void LLWU_GetVersionId ( LLWU_Type * base, llwu_version_id_t * versionId ) [inline], [static]`

This function gets the LLWU version ID, including the major version number, the minor version number, and the feature specification number.

Parameters

<code>base</code>	LLWU peripheral base address.
<code>versionId</code>	A pointer to the version ID structure.

### 17.8.2 static void LLWU\_GetParam ( **LLWU\_Type** \* *base*, **llwu\_param\_t** \* *param* ) [inline], [static]

This function gets the LLWU parameter, including a wakeup pin number, a module number, a DMA number, and a pin filter number.

Parameters

<i>base</i>	LLWU peripheral base address.
<i>param</i>	A pointer to the LLWU parameter structure.

### 17.8.3 void LLWU\_SetExternalWakePinMode ( LLWU\_Type \* *base*, uint32\_t *pinIndex*, llwu\_external\_pin\_mode\_t *pinMode* )

This function sets the external input pin source mode that is used as a wake up source.

Parameters

<i>base</i>	LLWU peripheral base address.
<i>pinIndex</i>	A pin index to be enabled as an external wakeup source starting from 1.
<i>pinMode</i>	A pin configuration mode defined in the llwu_external_pin_modes_t.

### 17.8.4 bool LLWU\_GetExternalWakePinFlag ( LLWU\_Type \* *base*, uint32\_t *pinIndex* )

This function checks the external pin flag to detect whether the MCU is woken up by the specific pin.

Parameters

<i>base</i>	LLWU peripheral base address.
<i>pinIndex</i>	A pin index, which starts from 1.

Returns

True if the specific pin is a wakeup source.

### 17.8.5 void LLWU\_ClearExternalWakePinFlag ( LLWU\_Type \* *base*, uint32\_t *pinIndex* )

This function clears the external wakeup source flag for a specific pin.

Parameters

<i>base</i>	LLWU peripheral base address.
<i>pinIndex</i>	A pin index, which starts from 1.

#### 17.8.6 static void LLWU\_EnableInternalModuleInterruptWakup ( LLWU\_Type \* *base*, uint32\_t *moduleIndex*, bool *enable* ) [inline], [static]

This function enables/disables the internal module source mode that is used as a wake up source.

Parameters

<i>base</i>	LLWU peripheral base address.
<i>moduleIndex</i>	A module index to be enabled as an internal wakeup source starting from 1.
<i>enable</i>	An enable or a disable setting

#### 17.8.7 static bool LLWU\_GetInternalWakeupModuleFlag ( LLWU\_Type \* *base*, uint32\_t *moduleIndex* ) [inline], [static]

This function checks the external pin flag to detect whether the system is woken up by the specific pin.

Parameters

<i>base</i>	LLWU peripheral base address.
<i>moduleIndex</i>	A module index, which starts from 1.

Returns

True if the specific pin is a wake up source.

#### 17.8.8 static void LLWU\_EnableInternalModuleDmaRequestWakup ( LLWU\_Type \* *base*, uint32\_t *moduleIndex*, bool *enable* ) [inline], [static]

This function enables/disables the internal DMA that is used as a wake up source.

Parameters

<i>base</i>	LLWU peripheral base address.
<i>moduleIndex</i>	An internal module index which is used as a DMA request source, starting from 1.
<i>enable</i>	Enable or disable the DMA request source

### 17.8.9 void LLWU\_SetPinFilterMode ( LLWU\_Type \* *base*, uint32\_t *filterIndex*, llwu\_external\_pin\_filter\_mode\_t *filterMode* )

This function sets the pin filter configuration.

Parameters

<i>base</i>	LLWU peripheral base address.
<i>filterIndex</i>	A pin filter index used to enable/disable the digital filter, starting from 1.
<i>filterMode</i>	A filter mode configuration

### 17.8.10 bool LLWU\_GetPinFilterFlag ( LLWU\_Type \* *base*, uint32\_t *filterIndex* )

This function gets the pin filter flag.

Parameters

<i>base</i>	LLWU peripheral base address.
<i>filterIndex</i>	A pin filter index, which starts from 1.

Returns

True if the flag is a source of the existing low-leakage power mode.

### 17.8.11 void LLWU\_ClearPinFilterFlag ( LLWU\_Type \* *base*, uint32\_t *filterIndex* )

This function clears the pin filter flag.

Parameters

<i>base</i>	LLWU peripheral base address.
<i>filterIndex</i>	A pin filter index to clear the flag, starting from 1.

### 17.8.12 void LLWU\_SetResetPinMode ( LLWU\_Type \* *base*, bool *pinEnable*, bool *pinFilterEnable* )

This function determines how the reset pin is used as a low leakage mode exit source.

Parameters

<i>base</i>	LLWU peripheral base address.
<i>pinEnable</i>	Enable reset the pin filter
<i>pinFilterEnable</i>	Specify whether the pin filter is enabled in Low-Leakage power mode.

# Chapter 18

## LPTMR: Low-Power Timer

### 18.1 Overview

The MCUXpresso SDK provides a driver for the Low-Power Timer (LPTMR) of MCUXpresso SDK devices.

### 18.2 Function groups

The LPTMR driver supports operating the module as a time counter or as a pulse counter.

#### 18.2.1 Initialization and deinitialization

The function [LPTMR\\_Init\(\)](#) initializes the LPTMR with specified configurations. The function [LPTMR\\_GetDefaultConfig\(\)](#) gets the default configurations. The initialization function configures the LPTMR for a timer or a pulse counter mode mode. It also sets up the LPTMR's free running mode operation and a clock source.

The function [LPTMR\\_DeInit\(\)](#) disables the LPTMR module and gates the module clock.

#### 18.2.2 Timer period Operations

The function [LPTMR\\_SetTimerPeriod\(\)](#) sets the timer period in units of count. Timers counts from 0 to the count value set here.

The function [LPTMR\\_GetCurrentTimerCount\(\)](#) reads the current timer counting value. This function returns the real-time timer counting value ranging from 0 to a timer period.

The timer period operation function takes the count value in ticks. Call the utility macros provided in the `fsl_common.h` file to convert to microseconds or milliseconds.

#### 18.2.3 Start and Stop timer operations

The function [LPTMR\\_StartTimer\(\)](#) starts the timer counting. After calling this function, the timer counts up to the counter value set earlier by using the [LPTMR\\_SetPeriod\(\)](#) function. Each time the timer reaches the count value and increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

The function [LPTMR\\_StopTimer\(\)](#) stops the timer counting and resets the timer's counter register.

## 18.2.4 Status

Provides functions to get and clear the LPTMR status.

## 18.2.5 Interrupt

Provides functions to enable/disable LPTMR interrupts and get the currently enabled interrupts.

## 18.3 Typical use case

### 18.3.1 LPTMR tick example

Updates the LPTMR period and toggles an LED periodically. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/lptmr

## Data Structures

- struct [lptmr\\_config\\_t](#)  
*LPTMR config structure.* [More...](#)

## Enumerations

- enum [lptmr\\_pin\\_select\\_t](#) {
   
   kLPTMR\_PinSelectInput\_0 = 0x0U,
   
   kLPTMR\_PinSelectInput\_1 = 0x1U,
   
   kLPTMR\_PinSelectInput\_2 = 0x2U,
   
   kLPTMR\_PinSelectInput\_3 = 0x3U }
   
*LPTMR pin selection used in pulse counter mode.*
- enum [lptmr\\_pin\\_polarity\\_t](#) {
   
   kLPTMR\_PinPolarityActiveHigh = 0x0U,
   
   kLPTMR\_PinPolarityActiveLow = 0x1U }
   
*LPTMR pin polarity used in pulse counter mode.*
- enum [lptmr\\_timer\\_mode\\_t](#) {
   
   kLPTMR\_TimerModeTimeCounter = 0x0U,
   
   kLPTMR\_TimerModePulseCounter = 0x1U }
   
*LPTMR timer mode selection.*
- enum [lptmr\\_prescaler\\_glitch\\_value\\_t](#) {

```

kLPTMR_Prescale_Glitch_0 = 0x0U,
kLPTMR_Prescale_Glitch_1 = 0x1U,
kLPTMR_Prescale_Glitch_2 = 0x2U,
kLPTMR_Prescale_Glitch_3 = 0x3U,
kLPTMR_Prescale_Glitch_4 = 0x4U,
kLPTMR_Prescale_Glitch_5 = 0x5U,
kLPTMR_Prescale_Glitch_6 = 0x6U,
kLPTMR_Prescale_Glitch_7 = 0x7U,
kLPTMR_Prescale_Glitch_8 = 0x8U,
kLPTMR_Prescale_Glitch_9 = 0x9U,
kLPTMR_Prescale_Glitch_10 = 0xAU,
kLPTMR_Prescale_Glitch_11 = 0xBU,
kLPTMR_Prescale_Glitch_12 = 0xCU,
kLPTMR_Prescale_Glitch_13 = 0xDU,
kLPTMR_Prescale_Glitch_14 = 0xEU,
kLPTMR_Prescale_Glitch_15 = 0xFU }

```

*LPTMR prescaler/glitch filter values.*

- enum `lptmr_prescaler_clock_select_t` {
   
kLPTMR\_PrescalerClock\_0 = 0x0U,
 kLPTMR\_PrescalerClock\_1 = 0x1U,
 kLPTMR\_PrescalerClock\_2 = 0x2U,
 kLPTMR\_PrescalerClock\_3 = 0x3U }

*LPTMR prescaler/glitch filter clock select.*

- enum `lptmr_interrupt_enable_t` { `kLPTMR_TimerInterruptEnable` = LPTMR\_CSR\_TIE\_MASK }
- List of the LPTMR interrupts.*
- enum `lptmr_status_flags_t` { `kLPTMR_TimerCompareFlag` = LPTMR\_CSR\_TCF\_MASK }
- List of the LPTMR status flags.*

## Driver version

- #define `FSL_LPTMR_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 1)`)  
*Version 2.1.1.*

## Initialization and deinitialization

- void `LPTMR_Init` (LPTMR\_Type \*base, const `lptmr_config_t` \*config)  
*Ungates the LPTMR clock and configures the peripheral for a basic operation.*
- void `LPTMR_Deinit` (LPTMR\_Type \*base)  
*Gates the LPTMR clock.*
- void `LPTMR_GetDefaultConfig` (`lptmr_config_t` \*config)  
*Fills in the LPTMR configuration structure with default settings.*

## Interrupt Interface

- static void `LPTMR_EnableInterrupts` (LPTMR\_Type \*base, uint32\_t mask)  
*Enables the selected LPTMR interrupts.*
- static void `LPTMR_DisableInterrupts` (LPTMR\_Type \*base, uint32\_t mask)  
*Disables the selected LPTMR interrupts.*

- static uint32\_t [LPTMR\\_GetEnabledInterrupts](#) (LPTMR\_Type \*base)  
*Gets the enabled LPTMR interrupts.*

## Status Interface

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

## Read and write the timer period

- static void [LPTMR\\_SetTimerPeriod](#) (LPTMR\_Type \*base, uint32\_t ticks)  
*Sets the timer period in units of count.*
- static uint32\_t [LPTMR\\_GetCurrentTimerCount](#) (LPTMR\_Type \*base)  
*Reads the current timer counting value.*

## Timer Start and Stop

- static void [LPTMR\\_StartTimer](#) (LPTMR\_Type \*base)  
*Starts the timer.*
- static void [LPTMR\\_StopTimer](#) (LPTMR\_Type \*base)  
*Stops the timer.*

## 18.4 Data Structure Documentation

### 18.4.1 struct lptmr\_config\_t

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

The configuration struct can be made constant so it resides in flash.

## Data Fields

- [lptmr\\_timer\\_mode\\_t timerMode](#)  
*Time counter mode or pulse counter mode.*
- [lptmr\\_pin\\_select\\_t pinSelect](#)  
*LPTMR pulse input pin select; used only in pulse counter mode.*
- [lptmr\\_pin\\_polarity\\_t pinPolarity](#)  
*LPTMR pulse input pin polarity; used only in pulse counter mode.*
- bool [enableFreeRunning](#)  
*True: enable free running, counter is reset on overflow False: counter is reset when the compare flag is set.*
- bool [bypassPrescaler](#)  
*True: bypass prescaler; false: use clock from prescaler.*
- [lptmr\\_prescaler\\_clock\\_select\\_t prescalerClockSource](#)

- [`lptmr\_prescaler\_glitch\_value\_t`](#) value  
*Prescaler or glitch filter value.*

## 18.5 Enumeration Type Documentation

### 18.5.1 enum lptmr\_pin\_select\_t

Enumerator

- `kLPTMR_PinSelectInput_0`** Pulse counter input 0 is selected.
- `kLPTMR_PinSelectInput_1`** Pulse counter input 1 is selected.
- `kLPTMR_PinSelectInput_2`** Pulse counter input 2 is selected.
- `kLPTMR_PinSelectInput_3`** Pulse counter input 3 is selected.

### 18.5.2 enum lptmr\_pin\_polarity\_t

Enumerator

- `kLPTMR_PinPolarityActiveHigh`** Pulse Counter input source is active-high.
- `kLPTMR_PinPolarityActiveLow`** Pulse Counter input source is active-low.

### 18.5.3 enum lptmr\_timer\_mode\_t

Enumerator

- `kLPTMR_TimerModeTimeCounter`** Time Counter mode.
- `kLPTMR_TimerModePulseCounter`** Pulse Counter mode.

### 18.5.4 enum lptmr\_prescaler\_glitch\_value\_t

Enumerator

- `kLPTMR_Prescale_Glitch_0`** Prescaler divide 2, glitch filter does not support this setting.
- `kLPTMR_Prescale_Glitch_1`** Prescaler divide 4, glitch filter 2.
- `kLPTMR_Prescale_Glitch_2`** Prescaler divide 8, glitch filter 4.
- `kLPTMR_Prescale_Glitch_3`** Prescaler divide 16, glitch filter 8.
- `kLPTMR_Prescale_Glitch_4`** Prescaler divide 32, glitch filter 16.
- `kLPTMR_Prescale_Glitch_5`** Prescaler divide 64, glitch filter 32.
- `kLPTMR_Prescale_Glitch_6`** Prescaler divide 128, glitch filter 64.
- `kLPTMR_Prescale_Glitch_7`** Prescaler divide 256, glitch filter 128.
- `kLPTMR_Prescale_Glitch_8`** Prescaler divide 512, glitch filter 256.

- kLPTMR\_Prescale\_Glitch\_9* Prescaler divide 1024, glitch filter 512.
- kLPTMR\_Prescale\_Glitch\_10* Prescaler divide 2048 glitch filter 1024.
- kLPTMR\_Prescale\_Glitch\_11* Prescaler divide 4096, glitch filter 2048.
- kLPTMR\_Prescale\_Glitch\_12* Prescaler divide 8192, glitch filter 4096.
- kLPTMR\_Prescale\_Glitch\_13* Prescaler divide 16384, glitch filter 8192.
- kLPTMR\_Prescale\_Glitch\_14* Prescaler divide 32768, glitch filter 16384.
- kLPTMR\_Prescale\_Glitch\_15* Prescaler divide 65536, glitch filter 32768.

### 18.5.5 enum lptmr\_prescaler\_clock\_select\_t

Note

Clock connections are SoC-specific

Enumerator

- kLPTMR\_PrescalerClock\_0* Prescaler/glitch filter clock 0 selected.
- kLPTMR\_PrescalerClock\_1* Prescaler/glitch filter clock 1 selected.
- kLPTMR\_PrescalerClock\_2* Prescaler/glitch filter clock 2 selected.
- kLPTMR\_PrescalerClock\_3* Prescaler/glitch filter clock 3 selected.

### 18.5.6 enum lptmr\_interrupt\_enable\_t

Enumerator

*kLPTMR\_TimerInterruptEnable* Timer interrupt enable.

### 18.5.7 enum lptmr\_status\_flags\_t

Enumerator

*kLPTMR\_TimerCompareFlag* Timer compare flag.

## 18.6 Function Documentation

### 18.6.1 void LPTMR\_Init ( LPTMR\_Type \* *base*, const lptmr\_config\_t \* *config* )

Note

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

Parameters

<i>base</i>	LPTMR peripheral base address
<i>config</i>	A pointer to the LPTMR configuration structure.

### 18.6.2 void LPTMR\_Deinit ( LPTMR\_Type \* *base* )

Parameters

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

### 18.6.3 void LPTMR\_GetDefaultConfig ( lptmr\_config\_t \* *config* )

The default values are as follows.

```
*     config->timerMode = kLPTMR_TimerModeTimeCounter;
*     config->pinSelect = kLPTMR_PinSelectInput_0;
*     config->pinPolarity = kLPTMR_PinPolarityActiveHigh;
*     config->enableFreeRunning = false;
*     config->bypassPrescaler = true;
*     config->prescalerClockSource = kLPTMR_PrescalerClock_1;
*     config->value = kLPTMR_Prescale_Glitch_0;
*
```

Parameters

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

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

Parameters

<i>base</i>	LPTMR peripheral base address
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">lptmr_interrupt_enable_t</a>

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

Parameters

<i>base</i>	LPTMR peripheral base address
<i>mask</i>	The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">lptmr_interrupt_enable_t</a> .

#### 18.6.6 static uint32\_t LPTMR\_GetEnabledInterrupts ( LPTMR\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

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

#### 18.6.7 static uint32\_t LPTMR\_GetStatusFlags ( LPTMR\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

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

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

Parameters

---

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

### 18.6.9 static void LPTMR\_SetTimerPeriod ( LPTMR\_Type \* *base*, uint32\_t *ticks* ) [inline], [static]

Timers counts from 0 until it equals the count value set here. The count value is written to the CMR register.

Note

1. The TCF flag is set with the CNR equals the count provided here and then increments.
2. Call the utility macros provided in the `fsl_common.h` to convert to ticks.

Parameters

<i>base</i>	LPTMR peripheral base address
<i>ticks</i>	A timer period in units of ticks, which should be equal or greater than 1.

### 18.6.10 static uint32\_t LPTMR\_GetCurrentTimerCount ( LPTMR\_Type \* *base* ) [inline], [static]

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note

Call the utility macros provided in the `fsl_common.h` to convert ticks to usec or msec.

Parameters

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

Returns

The current counter value in ticks

**18.6.11 static void LPTMR\_StartTimer ( LPTMR\_Type \* *base* ) [inline],  
[static]**

After calling this function, the timer counts up to the CMR register value. Each time the timer reaches the CMR value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

Parameters

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

**18.6.12 static void LPTMR\_StopTimer ( LPTMR\_Type \* *base* ) [inline],  
[static]**

This function stops the timer and resets the timer's counter register.

Parameters

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

# Chapter 19

## LPUART: Low Power Universal Asynchronous Receiver-/Transmitter Driver

### 19.1 Overview

#### Modules

- [LPUART DMA Driver](#)
- [LPUART Driver](#)
- [LPUART FreeRTOS Driver](#)

## 19.2 LPUART Driver

### 19.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power UART (LPUART) module of MCUXpresso SDK devices.

### 19.2.2 Typical use case

#### 19.2.2.1 LPUART Operation

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

## Data Structures

- struct [lpuart\\_config\\_t](#)  
*LPUART configuration structure. [More...](#)*
- struct [lpuart\\_transfer\\_t](#)  
*LPUART transfer structure. [More...](#)*
- struct [lpuart\\_handle\\_t](#)  
*LPUART handle structure. [More...](#)*

## Macros

- #define [UART\\_RETRY\\_TIMES](#) 0U /\* Defining to zero means to keep waiting for the flag until it is assert/deassert. \*/  
*Retry times for waiting flag.*

## Typedefs

- typedef void(\* [lpuart\\_transfer\\_callback\\_t](#))(LPUART\_Type \*base, lpuart\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*LPUART transfer callback function.*

## Enumerations

- enum {
   
kStatus\_LPUART\_TxBusy = MAKE\_STATUS(kStatusGroup\_LPUART, 0),
   
kStatus\_LPUART\_RxBusy = MAKE\_STATUS(kStatusGroup\_LPUART, 1),
   
kStatus\_LPUART\_TxIdle = MAKE\_STATUS(kStatusGroup\_LPUART, 2),
   
kStatus\_LPUART\_RxIdle = MAKE\_STATUS(kStatusGroup\_LPUART, 3),
   
kStatus\_LPUART\_TxWatermarkTooLarge = MAKE\_STATUS(kStatusGroup\_LPUART, 4),
   
kStatus\_LPUART\_RxWatermarkTooLarge = MAKE\_STATUS(kStatusGroup\_LPUART, 5),
   
kStatus\_LPUART\_FlagCannotClearManually = MAKE\_STATUS(kStatusGroup\_LPUART, 6),
   
kStatus\_LPUART\_Error = MAKE\_STATUS(kStatusGroup\_LPUART, 7),
   
kStatus\_LPUART\_RxRingBufferOverrun,
   
kStatus\_LPUART\_RxHardwareOverrun = MAKE\_STATUS(kStatusGroup\_LPUART, 9),
   
kStatus\_LPUART\_NoiseError = MAKE\_STATUS(kStatusGroup\_LPUART, 10),
   
kStatus\_LPUART\_FramingError = MAKE\_STATUS(kStatusGroup\_LPUART, 11),
   
kStatus\_LPUART\_ParityError = MAKE\_STATUS(kStatusGroup\_LPUART, 12),
   
kStatus\_LPUART\_BaudrateNotSupport,
   
kStatus\_LPUART\_IdleLineDetected = MAKE\_STATUS(kStatusGroup\_LPUART, 14),
   
kStatus\_LPUART\_Timeout = MAKE\_STATUS(kStatusGroup\_LPUART, 15) }

*Error codes for the LPUART driver.*

- enum `lpuart_parity_mode_t` {
   
kLPUART\_ParityDisabled = 0x0U,
   
kLPUART\_ParityEven = 0x2U,
   
kLPUART\_ParityOdd = 0x3U }
- LPUART parity mode.*
- enum `lpuart_data_bits_t` { kLPUART\_EightDataBits = 0x0U }
- LPUART data bits count.*
- enum `lpuart_stop_bit_count_t` {
   
kLPUART\_OneStopBit = 0U,
   
kLPUART\_TwoStopBit = 1U }
- LPUART stop bit count.*
- enum `lpuart_transmit_cts_source_t` {
   
kLPUART\_CtsSourcePin = 0U,
   
kLPUART\_CtsSourceMatchResult = 1U }
- LPUART transmit CTS source.*
- enum `lpuart_transmit_cts_config_t` {
   
kLPUART\_CtsSampleAtStart = 0U,
   
kLPUART\_CtsSampleAtIdle = 1U }
- LPUART transmit CTS configure.*
- enum `lpuart_idle_type_select_t` {
   
kLPUART\_IdleTypeStartBit = 0U,
   
kLPUART\_IdleTypeStopBit = 1U }
- LPUART idle flag type defines when the receiver starts counting.*
- enum `lpuart_idle_config_t` {

```
kLPUART_IdleCharacter1 = 0U,
kLPUART_IdleCharacter2 = 1U,
kLPUART_IdleCharacter4 = 2U,
kLPUART_IdleCharacter8 = 3U,
kLPUART_IdleCharacter16 = 4U,
kLPUART_IdleCharacter32 = 5U,
kLPUART_IdleCharacter64 = 6U,
kLPUART_IdleCharacter128 = 7U }
```

*LPUART idle detected configuration.*

- enum `_lpuart_interrupt_enable` {
 

```
kLPUART_LinBreakInterruptEnable = (LPUART_BAUD_LBKDIIE_MASK >> 8U),
kLPUART_RxActiveEdgeInterruptEnable = (LPUART_BAUD_RXEDGIE_MASK >> 8U),
kLPUART_TxDataRegEmptyInterruptEnable = (LPUART_CTRL_TIE_MASK),
kLPUART_TransmissionCompleteInterruptEnable = (LPUART_CTRL_TCIE_MASK),
kLPUART_RxDataRegFullInterruptEnable = (LPUART_CTRL_RIE_MASK),
kLPUART_IdleLineInterruptEnable = (LPUART_CTRL_ILIE_MASK),
kLPUART_RxOverrunInterruptEnable = (LPUART_CTRL_ORIE_MASK),
kLPUART_NoiseErrorInterruptEnable = (LPUART_CTRL_NEIE_MASK),
kLPUART_FramingErrorInterruptEnable = (LPUART_CTRL_FEIE_MASK),
kLPUART_ParityErrorInterruptEnable = (LPUART_CTRL_PEIE_MASK),
kLPUART_Match1InterruptEnable = (LPUART_CTRL_MA1IE_MASK),
kLPUART_Match2InterruptEnable = (LPUART_CTRL_MA2IE_MASK) }
```

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

- enum `_lpuart_flags` {
 

```
kLPUART_TxDataRegEmptyFlag,
kLPUART_TransmissionCompleteFlag,
kLPUART_RxDataRegFullFlag = (LPUART_STAT_RDRF_MASK),
kLPUART_IdleLineFlag = (LPUART_STAT_IDLE_MASK),
kLPUART_RxOverrunFlag = (LPUART_STAT_OR_MASK),
kLPUART_NoiseErrorFlag = (LPUART_STAT_NF_MASK),
kLPUART_FramingErrorFlag,
kLPUART_ParityErrorFlag = (LPUART_STAT_PF_MASK),
kLPUART_LinBreakFlag = (LPUART_STAT_LBKDIF_MASK),
kLPUART_RxActiveEdgeFlag = (LPUART_STAT_RXEDGIF_MASK),
kLPUART_RxActiveFlag,
kLPUART_DataMatch1Flag,
kLPUART_DataMatch2Flag }
```

*LPUART status flags.*

## Driver version

- #define `FSL_LPUART_DRIVER_VERSION` (`MAKE_VERSION(2, 5, 3)`)

*LPUART driver version.*

## Initialization and deinitialization

- **status\_t LPUART\_Init** (LPUART\_Type \*base, const lpuart\_config\_t \*config, uint32\_t srcClock\_Hz)

*Initializes an LPUART instance with the user configuration structure and the peripheral clock.*

- **void LPUART\_Deinit** (LPUART\_Type \*base)

*Deinitializes a LPUART instance.*

- **void LPUART\_GetDefaultConfig** (lpuart\_config\_t \*config)

*Gets the default configuration structure.*

## Module configuration

- **status\_t LPUART\_SetBaudRate** (LPUART\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)

*Sets the LPUART instance baudrate.*

- **void LPUART\_Enable9bitMode** (LPUART\_Type \*base, bool enable)

*Enable 9-bit data mode for LPUART.*

- **static void LPUART\_SetMatchAddress** (LPUART\_Type \*base, uint16\_t address1, uint16\_t address2)

*Set the LPUART address.*

- **static void LPUART\_EnableMatchAddress** (LPUART\_Type \*base, bool match1, bool match2)

*Enable the LPUART match address feature.*

## Status

- **uint32\_t LPUART\_GetStatusFlags** (LPUART\_Type \*base)

*Gets LPUART status flags.*

- **status\_t LPUART\_ClearStatusFlags** (LPUART\_Type \*base, uint32\_t mask)

*Clears status flags with a provided mask.*

## Interrupts

- **void LPUART\_EnableInterrupts** (LPUART\_Type \*base, uint32\_t mask)

*Enables LPUART interrupts according to a provided mask.*

- **void LPUART\_DisableInterrupts** (LPUART\_Type \*base, uint32\_t mask)

*Disables LPUART interrupts according to a provided mask.*

- **uint32\_t LPUART\_GetEnabledInterrupts** (LPUART\_Type \*base)

*Gets enabled LPUART interrupts.*

## DMA Configuration

- **static uint32\_t LPUART\_GetDataRegisterAddress** (LPUART\_Type \*base)

*Gets the LPUART data register address.*

- **static void LPUART\_EnableTxDMA** (LPUART\_Type \*base, bool enable)

*Enables or disables the LPUART transmitter DMA request.*

- static void [LPUART\\_EnableRxDMA](#) (LPUART\_Type \*base, bool enable)  
*Enables or disables the LPUART receiver DMA.*

## Bus Operations

- uint32\_t [LPUART\\_GetInstance](#) (LPUART\_Type \*base)  
*Get the LPUART instance from peripheral base address.*
- static void [LPUART\\_EnableTx](#) (LPUART\_Type \*base, bool enable)  
*Enables or disables the LPUART transmitter.*
- static void [LPUART\\_EnableRx](#) (LPUART\_Type \*base, bool enable)  
*Enables or disables the LPUART receiver.*
- static void [LPUART\\_WriteByte](#) (LPUART\_Type \*base, uint8\_t data)  
*Writes to the transmitter register.*
- static uint8\_t [LPUART\\_ReadByte](#) (LPUART\_Type \*base)  
*Reads the receiver register.*
- void [LPUART\\_SendAddress](#) (LPUART\_Type \*base, uint8\_t address)  
*Transmit an address frame in 9-bit data mode.*
- status\_t [LPUART\\_WriteBlocking](#) (LPUART\_Type \*base, const uint8\_t \*data, size\_t length)  
*Writes to the transmitter register using a blocking method.*
- status\_t [LPUART\\_ReadBlocking](#) (LPUART\_Type \*base, uint8\_t \*data, size\_t length)  
*Reads the receiver data register using a blocking method.*

## Transactional

- void [LPUART\\_TransferCreateHandle](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle, [lpuart\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the LPUART handle.*
- status\_t [LPUART\\_TransferSendNonBlocking](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle, [lpuart\\_transfer\\_t](#) \*xfer)  
*Transmits a buffer of data using the interrupt method.*
- void [LPUART\\_TransferStartRingBuffer](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle, uint8\_t \*ringBuffer, size\_t ringBufferSize)  
*Sets up the RX ring buffer.*
- void [LPUART\\_TransferStopRingBuffer](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle)  
*Aborts the background transfer and uninstalls the ring buffer.*
- size\_t [LPUART\\_TransferGetRxRingBufferLength](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle)  
*Get the length of received data in RX ring buffer.*
- void [LPUART\\_TransferAbortSend](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle)  
*Aborts the interrupt-driven data transmit.*
- status\_t [LPUART\\_TransferGetSendCount](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of bytes that have been sent out to bus.*
- status\_t [LPUART\\_TransferReceiveNonBlocking](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle, [lpuart\\_transfer\\_t](#) \*xfer, size\_t \*receivedBytes)  
*Receives a buffer of data using the interrupt method.*
- void [LPUART\\_TransferAbortReceive](#) (LPUART\_Type \*base, lpuart\_handle\_t \*handle)  
*Aborts the interrupt-driven data receiving.*

- **status\_t LPUART\_TransferGetReceiveCount** (LPUART\_Type \*base, lpuart\_handle\_t \*handle, uint32\_t \*count)

*Gets the number of bytes that have been received.*

- **void LPUART\_TransferHandleIRQ** (LPUART\_Type \*base, void \*irqHandle)

*LPUART IRQ handle function.*

- **void LPUART\_TransferHandleErrorIRQ** (LPUART\_Type \*base, void \*irqHandle)

*LPUART Error IRQ handle function.*

## 19.2.3 Data Structure Documentation

### 19.2.3.1 struct lpuart\_config\_t

#### Data Fields

- **uint32\_t baudRate\_Bps**  
*LPUART baud rate.*
- **lpuart\_parity\_mode\_t parityMode**  
*Parity mode, disabled (default), even, odd.*
- **lpuart\_data\_bits\_t dataBitsCount**  
*Data bits count, eight (default), seven.*
- **bool isMsb**  
*Data bits order, LSB (default), MSB.*
- **lpuart\_stop\_bit\_count\_t stopBitCount**  
*Number of stop bits, 1 stop bit (default) or 2 stop bits.*
- **bool enableRxRTS**  
*RX RTS enable.*
- **bool enableTxCTS**  
*TX CTS enable.*
- **lpuart\_transmit\_cts\_source\_t txCtsSource**  
*TX CTS source.*
- **lpuart\_transmit\_cts\_config\_t txCtsConfig**  
*TX CTS configure.*
- **lpuart\_idle\_type\_select\_t rxIdleType**  
*RX IDLE type.*
- **lpuart\_idle\_config\_t rxIdleConfig**  
*RX IDLE configuration.*
- **bool enableTx**  
*Enable TX.*
- **bool enableRx**  
*Enable RX.*

#### Field Documentation

- (1) **lpuart\_idle\_type\_select\_t lpuart\_config\_t::rxIdleType**
- (2) **lpuart\_idle\_config\_t lpuart\_config\_t::rxIdleConfig**

### 19.2.3.2 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`

### 19.2.3.3 struct \_lpuart\_handle

#### Data Fields

- `const uint8_t *volatile txData`  
*Address of remaining data to send.*
- `volatile size_t txDataSize`  
*Size of the remaining data to send.*
- `size_t txDataSizeAll`  
*Size of the data to send out.*
- `uint8_t *volatile rxData`  
*Address of remaining data to receive.*
- `volatile size_t rxDataSize`  
*Size of the remaining data to receive.*
- `size_t rxDataSizeAll`  
*Size of the data to receive.*
- `uint8_t * rxRingBuffer`  
*Start address of the receiver ring buffer.*
- `size_t rxRingBufferSize`  
*Size of the ring buffer.*
- `volatile uint16_t rxRingBufferHead`  
*Index for the driver to store received data into ring buffer.*
- `volatile uint16_t rxRingBufferTail`  
*Index for the user to get data from the ring buffer.*
- `lpuart_transfer_callback_t callback`  
*Callback function.*
- `void * userData`  
*LPUART callback function parameter.*

- volatile uint8\_t **txState**  
*TX transfer state.*
- volatile uint8\_t **rxState**  
*RX transfer state.*

### Field Documentation

- (1) const uint8\_t\* volatile lpuart\_handle\_t::txData
- (2) volatile size\_t lpuart\_handle\_t::txDataSize
- (3) size\_t lpuart\_handle\_t::txDataSizeAll
- (4) uint8\_t\* volatile lpuart\_handle\_t::rxData
- (5) volatile size\_t lpuart\_handle\_t::rxDataSize
- (6) size\_t lpuart\_handle\_t::rxDataSizeAll
- (7) uint8\_t\* lpuart\_handle\_t::rxRingBuffer
- (8) size\_t lpuart\_handle\_t::rxRingBufferSize
- (9) volatile uint16\_t lpuart\_handle\_t::rxRingBufferHead
- (10) volatile uint16\_t lpuart\_handle\_t::rxRingBufferTail
- (11) lpuart\_transfer\_callback\_t lpuart\_handle\_t::callback
- (12) void\* lpuart\_handle\_t::userData
- (13) volatile uint8\_t lpuart\_handle\_t::txState
- (14) volatile uint8\_t lpuart\_handle\_t::rxState

### 19.2.4 Macro Definition Documentation

19.2.4.1 #define FSL\_LPUART\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 3))

19.2.4.2 #define UART\_RETRY\_TIMES 0U /\* Defining to zero means to keep waiting for the flag until it is assert/deassert. \*/

### 19.2.5 Typedef Documentation

19.2.5.1 typedef void(\* lpuart\_transfer\_callback\_t)(LPUART\_Type \*base, lpuart\_handle\_t \*handle, status\_t status, void \*userData)

## 19.2.6 Enumeration Type Documentation

### 19.2.6.1 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.

### 19.2.6.2 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.

### 19.2.6.3 enum lpuart\_data\_bits\_t

Enumerator

*kLPUART\_EightDataBits* Eight data bit.

### 19.2.6.4 enum lpuart\_stop\_bit\_count\_t

Enumerator

*kLPUART\_OneStopBit* One stop bit.  
*kLPUART\_TwoStopBit* Two stop bits.

### 19.2.6.5 enum lpuart\_transmit\_cts\_source\_t

Enumerator

*kLPUART\_CtsSourcePin* CTS resource is the LPUART\_CTS pin.

*kLPUART\_CtsSourceMatchResult* CTS resource is the match result.

### 19.2.6.6 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.

### 19.2.6.7 enum lpuart\_idle\_type\_select\_t

Enumerator

*kLPUART\_IdleTypeStartBit* Start counting after a valid start bit.

*kLPUART\_IdleTypeStopBit* Start counting after a stop bit.

### 19.2.6.8 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.

### 19.2.6.9 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

#### 19.2.6.10 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

#### 19.2.7 Function Documentation

### 19.2.7.1 status\_t LPUART\_Init ( LPUART\_Type \* *base*, const lpuart\_config\_t \* *config*, uint32\_t *srcClock\_Hz* )

This function configures the LPUART module with user-defined settings. Call the [LPUART\\_GetDefaultConfig\(\)](#) function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
* lpuart_config_t lpuartConfig;
* lpuartConfig.baudRate_Bps = 115200U;
* lpuartConfig.parityMode = kLPUART_ParityDisabled;
* lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
* lpuartConfig.isMsb = false;
* lpuartConfig.stopBitCount = kLPUART_OneStopBit;
* lpuartConfig.txFifoWatermark = 0;
* lpuartConfig.rxFifoWatermark = 1;
* LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
*
```

Parameters

<i>base</i>	LPUART peripheral base address.
<i>config</i>	Pointer to a user-defined configuration structure.
<i>srcClock_Hz</i>	LPUART clock source frequency in HZ.

Return values

<i>kStatus_LPUART_BaudrateNotSupport</i>	Baudrate is not support in current clock source.
<i>kStatus_Success</i>	LPUART initialize succeed

### 19.2.7.2 void LPUART\_Deinit ( LPUART\_Type \* *base* )

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

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

### 19.2.7.3 void LPUART\_GetDefaultConfig ( lpuart\_config\_t \* *config* )

This function initializes the LPUART configuration structure to a default value. The default values are:  
*: lpuartConfig->baudRate\_Bps = 115200U; lpuartConfig->parityMode = kLPUART\_ParityDisabled;*  
*lpuartConfig->dataBitsCount = kLPUART\_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART\_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART\_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART\_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;*

Parameters

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

#### 19.2.7.4 status\_t LPUART\_SetBaudRate ( LPUART\_Type \* *base*, uint32\_t *baudRate\_Bps*, uint32\_t *srcClock\_Hz* )

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART\_Init.

```
* LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
*
```

Parameters

<i>base</i>	LPUART peripheral base address.
<i>baudRate_Bps</i>	LPUART baudrate to be set.
<i>srcClock_Hz</i>	LPUART clock source frequency in HZ.

Return values

<i>kStatus_LPUART_BaudrateNotSupport</i>	Baudrate is not supported in the current clock source.
<i>kStatus_Success</i>	Set baudrate succeeded.

#### 19.2.7.5 void LPUART\_Enable9bitMode ( LPUART\_Type \* *base*, bool *enable* )

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>enable</i>	true to enable, flase to disable.

#### 19.2.7.6 static void LPUART\_SetMatchAddress ( LPUART\_Type \* *base*, uint16\_t *address1*, uint16\_t *address2* ) [inline], [static]

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it

receives with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

#### Note

Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

#### Parameters

<i>base</i>	LPUART peripheral base address.
<i>address1</i>	LPUART slave address1.
<i>address2</i>	LPUART slave address2.

### 19.2.7.7 static void LPUART\_EnableMatchAddress ( LPUART\_Type \* *base*, bool *match1*, bool *match2* ) [inline], [static]

#### Parameters

<i>base</i>	LPUART peripheral base address.
<i>match1</i>	true to enable match address1, false to disable.
<i>match2</i>	true to enable match address2, false to disable.

### 19.2.7.8 uint32\_t LPUART\_GetStatusFlags ( LPUART\_Type \* *base* )

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators [\\_lpuart\\_flags](#). To check for a specific status, compare the return value with enumerators in the [\\_lpuart\\_flags](#). For example, to check whether the TX is empty:

```
*     if (kLPUART_TxDataRegEmptyFlag &
*         LPUART_GetStatusFlags(LPUART1))
*     {
*         ...
*     }
```

Parameters

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

Returns

LPUART status flags which are ORed by the enumerators in the \_lpuart\_flags.

### 19.2.7.9 **status\_t LPUART\_ClearStatusFlags ( LPUART\_Type \* *base*, uint32\_t *mask* )**

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only cleared or set by hardware are: kLPUART\_TxDataRegEmptyFlag, kLPUART\_TransmissionCompleteFlag, kLPUART\_RxDataRegFullFlag, kLPUART\_RxActiveFlag, kLPUART\_NoiseErrorFlag, kLPUART\_ParityErrorFlag, kLPUART\_TxFifoEmptyFlag, kLPUART\_RxFifoEmptyFlag Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>mask</i>	the status flags to be cleared. The user can use the enumerators in the _lpuart_status_flag_t to do the OR operation and get the mask.

Returns

0 succeed, others failed.

Return values

<i>kStatus_LPUART_FlagCannotClearManually</i>	The flag can't be cleared by this function but it is cleared automatically by hardware.
<i>kStatus_Success</i>	Status in the mask are cleared.

### 19.2.7.10 **void LPUART\_EnableInterrupts ( LPUART\_Type \* *base*, uint32\_t *mask* )**

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the [\\_lpuart\\_interrupt\\_enable](#). This examples shows how to enable TX empty interrupt and RX full interrupt:

```
*     LPUART_EnableInterrupts(LPUART1,
    kLPUART_TxDataRegEmptyInterruptEnable |
    kLPUART_RxDataRegFullInterruptEnable);
*
```

## Parameters

<i>base</i>	LPUART peripheral base address.
<i>mask</i>	The interrupts to enable. Logical OR of <a href="#">_lpuart_interrupt_enable</a> .

**19.2.7.11 void LPUART\_DisableInterrupts ( LPUART\_Type \* *base*, uint32\_t *mask* )**

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See [\\_lpuart\\_interrupt\\_enable](#). This example shows how to disable the TX empty interrupt and RX full interrupt:

```
*     LPUART_DisableInterrupts(LPUART1,
*                               kLPUART_TxDataRegEmptyInterruptEnable |
*                               kLPUART_RxDataRegFullInterruptEnable);
*
```

## Parameters

<i>base</i>	LPUART peripheral base address.
<i>mask</i>	The interrupts to disable. Logical OR of <a href="#">_lpuart_interrupt_enable</a> .

**19.2.7.12 uint32\_t LPUART\_GetEnabledInterrupts ( LPUART\_Type \* *base* )**

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators [\\_lpuart\\_interrupt\\_enable](#). To check a specific interrupt enable status, compare the return value with enumerators in [\\_lpuart\\_interrupt\\_enable](#). For example, to check whether the TX empty interrupt is enabled:

```
*     uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);
*
*     if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
*     {
*         ...
*     }
*
```

## Parameters

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

## Returns

LPUART interrupt flags which are logical OR of the enumerators in [\\_lpuart\\_interrupt\\_enable](#).

**19.2.7.13 static uint32\_t LPUART\_GetDataRegisterAddress ( LPUART\_Type \* *base* )  
[inline], [static]**

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

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

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

#### 19.2.7.14 static void LPUART\_EnableTxDMA ( LPUART\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>enable</i>	True to enable, false to disable.

#### 19.2.7.15 static void LPUART\_EnableRxDMA ( LPUART\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>enable</i>	True to enable, false to disable.

#### 19.2.7.16 uint32\_t LPUART\_GetInstance ( LPUART\_Type \* *base* )

Parameters

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

Returns

LPUART instance.

**19.2.7.17 static void LPUART\_EnableTx ( LPUART\_Type \* *base*, bool *enable* )  
[inline], [static]**

This function enables or disables the LPUART transmitter.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>enable</i>	True to enable, false to disable.

#### 19.2.7.18 static void LPUART\_EnableRx ( LPUART\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the LPUART receiver.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>enable</i>	True to enable, false to disable.

#### 19.2.7.19 static void LPUART\_WriteByte ( LPUART\_Type \* *base*, uint8\_t *data* ) [inline], [static]

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>data</i>	Data write to the TX register.

#### 19.2.7.20 static uint8\_t LPUART\_ReadByte ( LPUART\_Type \* *base* ) [inline], [static]

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

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

Returns

Data read from data register.

#### 19.2.7.21 void LPUART\_SendAddress ( LPUART\_Type \* *base*, uint8\_t *address* )

Parameters

<i>base</i>	LPUART peripheral base address.
<i>address</i>	LPUART slave address.

### 19.2.7.22 status\_t LPUART\_WriteBlocking ( **LPUART\_Type** \* *base*, **const uint8\_t** \* *data*, **size\_t** *length* )

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the dat to be sent out to the bus.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>data</i>	Start address of the data to write.
<i>length</i>	Size of the data to write.

Return values

<i>kStatus_LPUART_- Timeout</i>	Transmission timed out and was aborted.
<i>kStatus_Success</i>	Successfully wrote all data.

### 19.2.7.23 status\_t LPUART\_ReadBlocking ( **LPUART\_Type** \* *base*, **uint8\_t** \* *data*, **size\_t** *length* )

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>data</i>	Start address of the buffer to store the received data.
<i>length</i>	Size of the buffer.

Return values

<i>kStatus_LPUART_Rx-HardwareOverrun</i>	Receiver overrun happened while receiving data.
<i>kStatus_LPUART_Noise-Error</i>	Noise error happened while receiving data.
<i>kStatus_LPUART_FramingError</i>	Framing error happened while receiving data.
<i>kStatus_LPUART_Parity-Error</i>	Parity error happened while receiving data.
<i>kStatus_LPUART_Timeout</i>	Transmission timed out and was aborted.
<i>kStatus_Success</i>	Successfully received all data.

#### 19.2.7.24 void LPUART\_TransferCreateHandle ( **LPUART\_Type** \* *base*, **Ipuart\_handle\_t** \* *handle*, **Ipuart\_transfer\_callback\_t** *callback*, **void** \* *userData* )

This function initializes the LPUART handle, which can be used for other LPUART transactional APIs. Usually, for a specified LPUART instance, call this API once to get the initialized handle.

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the [LPUART\\_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as *ringBuffer*.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>callback</i>	Callback function.
<i>userData</i>	User data.

#### 19.2.7.25 **status\_t** LPUART\_TransferSendNonBlocking ( **LPUART\_Type** \* *base*, **Ipuart\_handle\_t** \* *handle*, **Ipuart\_transfer\_t** \* *xfer* )

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the [kStatus\\_LPUART\\_TxIdle](#) as status parameter.

## Note

The `kStatus_LPUART_TxIdle` is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the T-X, check the `kLPUART_TransmissionCompleteFlag` to ensure that the transmit is finished.

## Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>xfer</i>	LPUART transfer structure, see <a href="#">lpuart_transfer_t</a> .

## Return values

<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.

### 19.2.7.26 void LPUART\_TransferStartRingBuffer ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle*, uint8\_t \* *ringBuffer*, size\_t *ringBufferSize* )

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the [UART\\_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

## Note

When using RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, then only 31 bytes are used for saving data.

## Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.

<i>ringBuffer</i>	Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
<i>ringBufferSize</i>	size of the ring buffer.

#### 19.2.7.27 void LPUART\_TransferStopRingBuffer ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle* )

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.

#### 19.2.7.28 size\_t LPUART\_TransferGetRxRingBufferLength ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle* )

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.

Returns

Length of received data in RX ring buffer.

#### 19.2.7.29 void LPUART\_TransferAbortSend ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle* )

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are not sent out.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.

### 19.2.7.30 status\_t LPUART\_TransferGetSendCount ( **LPUART\_Type** \* *base*,                   **Ipuart\_Handle\_t** \* *handle*, **uint32\_t** \* *count* )

This function gets the number of bytes that have been sent out to bus by an interrupt method.

## Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>count</i>	Send bytes count.

## Return values

<i>kStatus_NoTransferInProgress</i>	No send in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <i>count</i> ;

**19.2.7.31 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 <a href="#">uart_transfer_t</a> .
<i>receivedBytes</i>	Bytes received from the ring buffer directly.

## Return values

<i>kStatus_Success</i>	Successfully queue the transfer into the transmit queue.
<i>kStatus_LPUART_Rx-Busy</i>	Previous receive request is not finished.
<i>kStatus_InvalidArgument</i>	Invalid argument.

### 19.2.7.32 void LPUART\_TransferAbortReceive ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle* )

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.

### 19.2.7.33 status\_t LPUART\_TransferGetReceiveCount ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been received.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>count</i>	Receive bytes count.

Return values

<i>kStatus_NoTransferIn-Progress</i>	No receive in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <i>count</i> ;

### 19.2.7.34 void LPUART\_TransferHandleIRQ ( LPUART\_Type \* *base*, void \* *irqHandle* )

This function handles the LPUART transmit and receive IRQ request.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>irqHandle</i>	LPUART handle pointer.

#### 19.2.7.35 void LPUART\_TransferHandleErrorIRQ ( LPUART\_Type \* *base*, void \* *irqHandle* )

This function handles the LPUART error IRQ request.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>irqHandle</i>	LPUART handle pointer.

## 19.3 LPUART DMA Driver

### 19.3.1 Overview

#### Data Structures

- struct [lpuart\\_dma\\_handle\\_t](#)  
*LPUART DMA handle. [More...](#)*

#### Typedefs

- [typedef void\(\\* lpuart\\_dma\\_transfer\\_callback\\_t \)](#)(LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*LPUART transfer callback function.*

#### Driver version

- [#define FSL\\_LPUART\\_DMA\\_DRIVER\\_VERSION \(MAKE\\_VERSION\(2, 5, 2\)\)](#)  
*LPUART DMA driver version.*

#### EDMA transactional

- [void LPUART\\_TransferCreateHandleDMA](#) (LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle, [lpuart\\_dma\\_transfer\\_callback\\_t](#) callback, void \*userData, [dma\\_handle\\_t](#) \*txDmaHandle, [dma\\_handle\\_t](#) \*rxDmaHandle)  
*Initializes the LPUART handle which is used in transactional functions.*
- [status\\_t LPUART\\_TransferSendDMA](#) (LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle, [lpuart\\_transfer\\_t](#) \*xfer)  
*Sends data using DMA.*
- [status\\_t LPUART\\_TransferReceiveDMA](#) (LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle, [lpuart\\_transfer\\_t](#) \*xfer)  
*Receives data using DMA.*
- [void LPUART\\_TransferAbortSendDMA](#) (LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle)  
*Aborts the sent data using DMA.*
- [void LPUART\\_TransferAbortReceiveDMA](#) (LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle)  
*Aborts the received data using DMA.*
- [status\\_t LPUART\\_TransferGetSendCountDMA](#) (LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of bytes written to the LPUART TX register.*
- [status\\_t LPUART\\_TransferGetReceiveCountDMA](#) (LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of received bytes.*
- [void LPUART\\_TransferDMAHandleIRQ](#) (LPUART\_Type \*base, void \*lpuartDmaHandle)  
*LPUART DMA IRQ handle function.*

## 19.3.2 Data Structure Documentation

### 19.3.2.1 struct \_lpuart\_dma\_handle

#### Data Fields

- **lpuart\_dma\_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.*
- **dma\_handle\_t \*txDmaHandle**  
*The DMA TX channel used.*
- **dma\_handle\_t \*rxDmaHandle**  
*The DMA RX channel used.*
- **volatile uint8\_t txState**  
*TX transfer state.*
- **volatile uint8\_t rxState**  
*RX transfer state.*

#### Field Documentation

- (1) **lpuart\_dma\_transfer\_callback\_t lpuart\_dma\_handle\_t::callback**
- (2) **void\* lpuart\_dma\_handle\_t::userData**
- (3) **size\_t lpuart\_dma\_handle\_t::rxDataSizeAll**
- (4) **size\_t lpuart\_dma\_handle\_t::txDataSizeAll**
- (5) **dma\_handle\_t\* lpuart\_dma\_handle\_t::txDmaHandle**
- (6) **dma\_handle\_t\* lpuart\_dma\_handle\_t::rxDmaHandle**
- (7) **volatile uint8\_t lpuart\_dma\_handle\_t::txState**

## 19.3.3 Macro Definition Documentation

### 19.3.3.1 #define FSL\_LPUART\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 2))

## 19.3.4 Typedef Documentation

### 19.3.4.1 **typedef void(\* lpuart\_dma\_transfer\_callback\_t)(LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle, status\_t status, void \*userData)**

### 19.3.5 Function Documentation

**19.3.5.1 void LPUART\_TransferCreateHandleDMA ( *LPUART\_Type* \* *base*,  
*lpuart\_dma\_handle\_t* \* *handle*, *lpuart\_dma\_transfer\_callback\_t* *callback*, *void* \*  
*userData*, *dma\_handle\_t* \* *txDmaHandle*, *dma\_handle\_t* \* *rxDmaHandle* )**

Note

This function disables all LPUART interrupts.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	Pointer to <i>lpuart_dma_handle_t</i> structure.
<i>callback</i>	Callback function.
<i>userData</i>	User data.
<i>txDmaHandle</i>	User-requested DMA handle for TX DMA transfer.
<i>rxDmaHandle</i>	User-requested DMA handle for RX DMA transfer.

**19.3.5.2 status\_t LPUART\_TransferSendDMA ( *LPUART\_Type* \* *base*,  
*lpuart\_dma\_handle\_t* \* *handle*, *lpuart\_transfer\_t* \* *xfer* )**

This function sends data using DMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>xfer</i>	LPUART DMA 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.
--------------------------------	-------------------

### 19.3.5.3 **status\_t LPUART\_TransferReceiveDMA ( LPUART\_Type \* *base*, lpuart\_dma\_handle\_t \* *handle*, lpuart\_transfer\_t \* *xfer* )**

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	Pointer to lpuart_dma_handle_t structure.
<i>xfer</i>	LPUART DMA transfer structure. See <a href="#">lpuart_transfer_t</a> .

Return values

<i>kStatus_Success</i>	if succeed, others failed.
<i>kStatus_LPUART_Rx-Busy</i>	Previous transfer on going.
<i>kStatus_InvalidArgument</i>	Invalid argument.

### 19.3.5.4 **void LPUART\_TransferAbortSendDMA ( LPUART\_Type \* *base*, lpuart\_dma\_handle\_t \* *handle* )**

This function aborts send data using DMA.

Parameters

<i>base</i>	LPUART peripheral base address
<i>handle</i>	Pointer to lpuart_dma_handle_t structure

### 19.3.5.5 **void LPUART\_TransferAbortReceiveDMA ( LPUART\_Type \* *base*, lpuart\_dma\_handle\_t \* *handle* )**

This function aborts the received data using DMA.

Parameters

<i>base</i>	LPUART peripheral base address
<i>handle</i>	Pointer to lpuart_dma_handle_t structure

#### 19.3.5.6 status\_t LPUART\_TransferGetSendCountDMA ( LPUART\_Type \* *base*, lpuart\_dma\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been written to 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> ;

#### 19.3.5.7 status\_t LPUART\_TransferGetReceiveCountDMA ( LPUART\_Type \* *base*, lpuart\_dma\_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;

#### 19.3.5.8 void LPUART\_TransferDMAHandleIRQ ( **LPUART\_Type** \* *base*, **void** \* *lpuartDmaHandle* )

This function handles the LPUART tx complete IRQ request and invoke user callback.

##### 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>lpuartDmaHandle</i>	LPUART handle pointer.

## 19.4 LPUART FreeRTOS Driver

### 19.4.1 Overview

#### Data Structures

- struct `lpuart_rtos_config_t`  
*LPUART RTOS configuration structure.* [More...](#)

#### Driver version

- #define `FSL_LPUART_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 6, 0)`)  
*LPUART FreeRTOS driver version.*

#### LPUART RTOS Operation

- int `LPUART_RTOS_Init` (`lpuart_rtos_handle_t *handle, lpuart_handle_t *t_handle, const lpuart_rtos_config_t *cfg`)  
*Initializes an LPUART instance for operation in RTOS.*
- int `LPUART_RTOS_Deinit` (`lpuart_rtos_handle_t *handle`)  
*Deinitializes an LPUART instance for operation.*

#### LPUART transactional Operation

- int `LPUART_RTOS_Send` (`lpuart_rtos_handle_t *handle, uint8_t *buffer, uint32_t length`)  
*Sends data in the background.*
- int `LPUART_RTOS_Receive` (`lpuart_rtos_handle_t *handle, uint8_t *buffer, uint32_t length, size_t *received`)  
*Receives data.*
- int `LPUART_RTOS_SetRxTimeout` (`lpuart_rtos_handle_t *handle, uint32_t rx_timeout_constant_ms, uint32_t rx_timeout_multiplier_ms`)  
*Set RX timeout in runtime.*
- int `LPUART_RTOS_SetTxTimeout` (`lpuart_rtos_handle_t *handle, uint32_t tx_timeout_constant_ms, uint32_t tx_timeout_multiplier_ms`)  
*Set TX timeout in runtime.*

### 19.4.2 Data Structure Documentation

#### 19.4.2.1 struct `lpuart_rtos_config_t`

##### Data Fields

- `LPUART_Type * base`  
*UART base address.*

- `uint32_t srclk`  
*UART source clock in Hz.*
- `uint32_t baudrate`  
*Desired communication speed.*
- `lpuart_parity_mode_t parity`  
*Parity setting.*
- `lpuart_stop_bit_count_t stopbits`  
*Number of stop bits to use.*
- `uint8_t * buffer`  
*Buffer for background reception.*
- `uint32_t buffer_size`  
*Size of buffer for background reception.*
- `uint32_t rx_timeout_constant_ms`  
*RX timeout applied per receive.*
- `uint32_t rx_timeout_multiplier_ms`  
*RX timeout added for each byte of the receive.*
- `uint32_t tx_timeout_constant_ms`  
*TX timeout applied per transmission.*
- `uint32_t tx_timeout_multiplier_ms`  
*TX timeout added for each byte of the transmission.*
- `bool enableRxRTS`  
*RX RTS enable.*
- `bool enableTxCTS`  
*TX CTS enable.*
- `lpuart_transmit_cts_source_t txCtsSource`  
*TX CTS source.*
- `lpuart_transmit_cts_config_t txCtsConfig`  
*TX CTS configure.*

## Field Documentation

- (1) `uint32_t lpuart_rtos_config_t::rx_timeout_multiplier_ms`
- (2) `uint32_t lpuart_rtos_config_t::tx_timeout_multiplier_ms`

## 19.4.3 Macro Definition Documentation

**19.4.3.1 #define FSL\_LPUART\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 6, 0))**

## 19.4.4 Function Documentation

**19.4.4.1 int LPUART\_RTOs\_Init ( `lpuart_rtos_handle_t * handle, lpuart_handle_t * t_handle, const lpuart_rtos_config_t * cfg`  )**

Parameters

<i>handle</i>	The RTOS LPUART handle, the pointer to an allocated space for RTOS context.
<i>t_handle</i>	The pointer to an allocated space to store the transactional layer internal state.
<i>cfg</i>	The pointer to the parameters required to configure the LPUART after initialization.

Returns

0 succeed, others failed

#### 19.4.4.2 int LPUART\_RTOS\_Deinit ( *Ipuart\_rtos\_handle\_t \* handle* )

This function deinitializes the LPUART module, sets all register value to the reset value, and releases the resources.

Parameters

<i>handle</i>	The RTOS LPUART handle.
---------------	-------------------------

#### 19.4.4.3 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.

#### 19.4.4.4 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.

#### 19.4.4.5 int LPUART\_RTOSETXTIMEOUT ( Ipuart\_rtos\_handle\_t \* *handle*, uint32\_t *rx\_timeout\_constant\_ms*, uint32\_t *rx\_timeout\_multiplier\_ms* )

This function can modify RX timeout between initialization and receive.

param handle The RTOS LPUART handle. param rx\_timeout\_constant\_ms RX timeout applied per receive. param rx\_timeout\_multiplier\_ms RX timeout added for each byte of the receive.

#### 19.4.4.6 int LPUART\_RTOSETXTIMEOUT ( Ipuart\_rtos\_handle\_t \* *handle*, uint32\_t *tx\_timeout\_constant\_ms*, uint32\_t *tx\_timeout\_multiplier\_ms* )

This function can modify TX timeout between initialization and send.

param handle The RTOS LPUART handle. param tx\_timeout\_constant\_ms TX timeout applied per transmission. param tx\_timeout\_multiplier\_ms TX timeout added for each byte of the transmission.

# Chapter 20

## MMAU: Memory Mapped Arithmetic Unit

### 20.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Memory Mapped Arithmetic Unit (MMAU) block of MCUXpresso SDK devices.

The Memory Mapped Arithmetic Unit (MMAU) provides acceleration to a set of math operations, including signed/unsigned multiplication and accumulation, division and root-square, and so on.

### 20.2 Function groups

#### 20.2.1 MMAU Initialization

To initialize the MMAU driver, call the `MMAU_EnableDMA(MMAU, true)` and `MMAU_EnableSupervisorOnly(MMAU, false)` functions. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/mmau`

#### 20.2.2 MMAU Interrupts

MMAU supports three interrupts: Accumulation Overflow (Q), Divide/Multiply Overflow (V) and Divide-by-Zero (DZ). These interrupts were definition at `mmau_flag_t` structure. The MMAU driver supports enable/disable the interrupts, get/clear the interrupt flags.

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

#### 20.2.3 MMAU Instruction flags

The MMAU driver provides four instruction flags: Accumulation Overflow (Q), Divide or Multiply Overflow (V), Divide-by-Zero (DZ) and Signed calculation result is negative (N). These flags were updated after each calculation. MMAU driver contains get and set functions to access instruction flags. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/mmau`

#### 20.2.4 MMAU Operators

The MMAU driver supports multiply, divide and square root functions. Each kind of function supports Unsigned Integer, Signed Integer (except square root) and Fractional Number as operator.

## 20.3 Typical use case and example

This example demonstrates the SIN calculation. Refer to the driver examples codes located at <SDK\_R-OOT>/boards/<BOARD>/driver\_examples/mmau

### Enumerations

- enum `mmau_interrupt_enable_t` {
   
`kMMAU_AccumOverflowInterruptEnable` = (`MMAU_CSR_QIE_MASK`),  
`kMMAU_OverflowInterruptEnable` = (`MMAU_CSR_VIE_MASK`),  
`kMMAU_DivideByZeroInterruptEnable` = (`MMAU_CSR_DZIE_MASK`) }
   
*MMAU interrupt configuration structure, default settings all disabled.*
- enum `mmau_interrupt_flag_t` {
   
`kMMAU_AccumOverflowInterruptFlag` = (`MMAU_CSR_QIF_MASK`),  
`kMMAU_OverflowInterruptFlag` = (`MMAU_CSR_VIF_MASK`),  
`kMMAU_DivideByZeroInterruptFlag` = (`MMAU_CSR_DZIF_MASK`) }
   
*MMAU interrupt and instruction flags.*
- enum `mmau_instruction_flag_t` {
   
`kMMAU_AccumOverflowInstructionFlag` = (`MMAU_CSR_Q_MASK`),  
`kMMAU_OverflowInstructionFlag` = (`MMAU_CSR_V_MASK`),  
`kMMAU_DivideByZeroInstructionFlag` = (`MMAU_CSR_DZ_MASK`),  
`kMMAU_NegativeInstructionFlag` = (`MMAU_CSR_N_MASK`) }
   
*MMAU interrupt and instruction flags.*

### Functions

- static `uint32_t MMAU_GetHwRevCmd (MMAU_Type *base)`  
*Gets hardware revision level.*

### Driver version

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

### Registers decorated load/store addresses

- #define `MMAU_X0` 0xF0004000UL  
*Accumulator register X0.*
- #define `MMAU_X1` 0xF0004004UL  
*Accumulator register X1.*
- #define `MMAU_X2` 0xF0004008UL  
*Accumulator register X2.*
- #define `MMAU_X3` 0xF000400CUL  
*Accumulator register X3.*
- #define `MMAU_A0` 0xF0004010UL  
*Accumulator register A0.*
- #define `MMAU_A1` 0xF0004014UL  
*Accumulator register A1.*
- #define `MMAU_A10` 0xF0004010UL  
*Accumulator register pair A10.*

## Unsigned integer instructions decorated load/store addresses

- #define **MMAU\_\_REGRW** 0xF0004000UL  
*Registers RW.*
- #define **MMAU\_\_UMUL** 0xF0004020UL  
*A10=X2\*X3.*
- #define **MMAU\_\_UMULD** 0xF0004040UL  
*A10=X2I\*X3.*
- #define **MMAU\_\_UMULDA** 0xF0004060UL  
*A10=A10\*X3.*
- #define **MMAU\_\_UMAC** 0xF00040A0UL  
*A10=X2\*X3+A10.*
- #define **MMAU\_\_UMACD** 0xF00040C0UL  
*A10=X2I\*X3+A10.*
- #define **MMAU\_\_UMACDA** 0xF00040E0UL  
*A10=A10\*X3+X2I.*
- #define **MMAU\_\_UDIV** 0xF0004120UL  
*X2I/X3=A10.*
- #define **MMAU\_\_UDIVD** 0xF0004140UL  
*A10=X2/X3.*
- #define **MMAU\_\_UDIVDA** 0xF0004160UL  
*A10=X2I/X3.*
- #define **MMAU\_\_UDIVDD** 0xF0004180UL  
*A10=A10/X3.*
- #define **MMAU\_\_UDIVDDA** 0xF00041A0UL  
*A10=A10/X32.*
- #define **MMAU\_\_USQR** 0xF0004220UL  
*A10=SQR(X3)*
- #define **MMAU\_\_USQRD** 0xF0004240UL  
*A10=SQR(X32)*
- #define **MMAU\_\_USQRDA** 0xF0004260UL  
*A10=SQR(A10)*

## Signed fractional instructions decorated load/store addresses

- #define **MMAU\_\_QSQR** 0xF00042A0UL  
*A10=SQR(X3)*
- #define **MMAU\_\_QSQRD** 0xF00042C0UL  
*A10=SQR(X32)*
- #define **MMAU\_\_QSQRDA** 0xF00042E0UL  
*A10=SQR(A10)*
- #define **MMAU\_\_QDIV** 0xF0004320UL  
*A10=X2/X3.*
- #define **MMAU\_\_QDIVD** 0xF0004340UL  
*A10=X2I/X3.*
- #define **MMAU\_\_QDIVDA** 0xF0004360UL  
*A10=A10/X3.*
- #define **MMAU\_\_QMUL** 0xF0004420UL  
*A10=X2\*X3.*
- #define **MMAU\_\_QMULD** 0xF0004440UL  
*A10=X2I\*X3.*
- #define **MMAU\_\_QMULDA** 0xF0004460UL

- $A10 = A10 * X3.$
- $\#define \text{MMAU\_QMAC} 0xF00044A0UL$   
 $A10 = X2 * X3 + A10.$
- $\#define \text{MMAU\_QMACD} 0xF00044C0UL$   
 $A10 = X21 * X3 + A10.$
- $\#define \text{MMAU\_QMACDA} 0xF00044E0UL$   
 $A10 = A10 * X3 + X21.$

## Signed integer instructions decorated load/store addresses

- $\#define \text{MMAU\_SMUL} 0xF0004620UL$   
 $A10 = X2 * X3.$
- $\#define \text{MMAU\_SMULD} 0xF0004640UL$   
 $A10 = X21 * X3.$
- $\#define \text{MMAU\_SMULDA} 0xF0004660UL$   
 $A10 = A10 * X3.$
- $\#define \text{MMAU\_SMAC} 0xF00046A0UL$   
 $A10 = X2 * X3 + A10.$
- $\#define \text{MMAU\_SMACD} 0xF00046C0UL$   
 $A10 = X21 * X3 + A10.$
- $\#define \text{MMAU\_SMACDA} 0xF00046E0UL$   
 $A10 = A10 * X3 + X21.$
- $\#define \text{MMAU\_SDIV} 0xF0004720UL$   
 $A10 = X2 / X3.$
- $\#define \text{MMAU\_SDIVD} 0xF0004740UL$   
 $A10 = X21 / X3.$
- $\#define \text{MMAU\_SDIVDA} 0xF0004760UL$   
 $A10 = A10 / X3.$
- $\#define \text{MMAU\_SDIVDD} 0xF0004780UL$   
 $A10 = X10 / X32.$
- $\#define \text{MMAU\_SDIVDDA} 0xF00047A0UL$   
 $A10 = A10 / X32.$

## Auxiliary decorated load/store addresses

- $\#define \text{MMAU\_SAT} 0xF0004800UL$   
*Saturation.*

## Fractional Data Type

- `typedef short int frac16_t`  
*Q0.15 fractional.*
- `typedef long frac24_t`  
*Q8.23 fractional.*
- `typedef long frac32_t`  
*Q0.31 fractional.*
- `typedef long long frac48_t`  
*Q16.47 fractional.*
- `typedef long long frac64_t`  
*Q0.63 fractional.*

## MMAU Configure

- static void **MMAU\_EnableDMA** (MMAU\_Type \*base, bool enable)  
*Enable DMA for MMAU module.*
- static void **MMAU\_EnableSupervisorOnly** (MMAU\_Type \*base, bool enable)  
*Enable supervisor only for MMAU module.*
- void **MMAU\_Reset** (MMAU\_Type \*base)  
*Set control/status register into reset state.*

## MMAU Interrupt

- static void **MMAU\_EnableInterrupts** (MMAU\_Type \*base, uint32\_t mask)  
*Enable MMAU interrupts.*
- static void **MMAU\_DisableInterrupts** (MMAU\_Type \*base, uint32\_t mask)  
*Disable MMAU interrupts.*
- static uint32\_t **MMAU\_GetEnabledInterrupts** (MMAU\_Type \*base)  
*Gets enabled interrupts.*
- static uint32\_t **MMAU\_GetInterruptFlags** (MMAU\_Type \*base)  
*Get interrupt flags.*
- void **MMAU\_ClearInterruptFlags** (MMAU\_Type \*base, uint32\_t mask)  
*Clears interrupt flags.*

## MMAU Instruction flag

- static uint32\_t **MMAU\_GetInstructionFlags** (MMAU\_Type \*base)  
*Gets the instruction flags.*
- void **MMAU\_SetInstructionFlags** (MMAU\_Type \*base, uint32\_t mask)  
*Sets the instruction flags.*
- void **MMAU\_ClearInstructionFlags** (MMAU\_Type \*base, uint32\_t mask)  
*Clears instruction flags.*

## Unsigned Integer Operands

- static void **MMAU\_ulta\_d** (register uint64\_t dval)  
*Load A10 accumulator register of the MMAU by 64-bit unsigned value.*
- static uint64\_t **MMAU\_d\_umul\_ll** (register uint32\_t lval1, register uint32\_t lval2)  
*Multiply two 32-bit unsigned values returning a 64-bit unsigned product.*
- static uint64\_t **MMAU\_d\_umul\_dl** (register uint64\_t dval, register uint32\_t lval)  
*Multiply 64-bit unsigned value with 32-bit unsigned value returning a 64-bit unsigned product.*
- static uint64\_t **MMAU\_d\_umuls\_dl** (register uint64\_t dval, register uint32\_t lval)  
*Saturating multiply 64-bit unsigned value with 32-bit unsigned value returning saturated 64-bit unsigned product.*
- static uint64\_t **MMAU\_d\_umula\_l** (register uint32\_t lval)  
*Multiply 32-bit unsigned value with 64-bit unsigned value stored in the A10 register of the MMAU returning a 64-bit unsigned product.*
- static uint64\_t **MMAU\_d\_umulas\_l** (register uint32\_t lval)  
*Saturating multiply 32-bit unsigned value with 64-bit unsigned value stored in the A10 register of the MMAU returning saturated 64-bit unsigned product.*
- static uint64\_t **MMAU\_d\_umac\_ll** (register uint32\_t lval1, register uint32\_t lval2)  
*Multiply two 32-bit unsigned values and add product with value stored in the A10 register of the MMAU returning a 64-bit unsigned A10 register value.*

- static uint64\_t **MMAU\_d\_umacs\_ll** (register uint32\_t lval1, register uint32\_t lval2)
 

*Saturating multiply two 32-bit unsigned values and add product with value stored in the A10 register of the MMAU returning a 64-bit unsigned A10 register value.*
- static uint64\_t **MMAU\_d\_umac\_dl** (register uint64\_t dval, register uint32\_t lval)
 

*Multiply 64-bit unsigned value with 32-bit unsigned value and add product with value stored in the A10 register of the MMAU returning a 64-bit unsigned A10 register value.*
- static uint64\_t **MMAU\_d\_umacs\_dl** (register uint64\_t dval, register uint32\_t lval)
 

*Saturating multiply 64-bit unsigned value with 32-bit unsigned value and add product with value stored in the A10 register of the MMAU returning saturated 64-bit unsigned A10 register value.*
- static uint64\_t **MMAU\_d\_umaca\_dl** (register uint64\_t dval, register uint32\_t lval)
 

*Multiply 32-bit unsigned value by value stored in the A10 register of the MMAU and add product with 64-bit unsigned value returning a 64-bit unsigned A10 register value.*
- static uint64\_t **MMAU\_d\_umacas\_dl** (register uint64\_t dval, register uint32\_t lval)
 

*Saturating multiply 32-bit unsigned value by value stored in the A10 register of the MMAU and add product with 64-bit unsigned value returning a saturated 64-bit unsigned A10 register value.*
- static uint32\_t **MMAU\_1\_udiv\_ll** (register uint32\_t lnum, register uint32\_t lden)
 

*Divide two 32-bit unsigned values returning a 32-bit unsigned quotient.*
- static uint64\_t **MMAU\_d\_udiv\_dl** (register uint64\_t dnum, register uint32\_t lden)
 

*Divide 64-bit unsigned value by 32-bit unsigned value returning a 64-bit unsigned quotient.*
- static uint64\_t **MMAU\_d\_udiv\_dd** (register uint64\_t dnum, register uint64\_tdden)
 

*Divide two 64-bit unsigned values returning a 64-bit unsigned quotient.*
- static uint64\_t **MMAU\_d\_udiva\_1** (register uint32\_t lden1)
 

*Divide 32-bit unsigned value stored in the A10 register of the MMAU by 32-bit unsigned value returning a 64-bit unsigned quotient.*
- static uint64\_t **MMAU\_d\_udiva\_d** (register uint64\_tdden1)
 

*Divide 64-bit unsigned value stored in the A10 register of the MMAU by 64-bit unsigned value returning a 64-bit unsigned quotient.*
- static uint32\_t **MMAU\_1\_usqr\_1** (register uint32\_t lrad)
 

*Compute and return a 32-bit unsigned square root of the 32-bit unsigned radicand.*
- static uint32\_t **MMAU\_1\_usqr\_d** (register uint64\_t drad)
 

*Compute and return a 32-bit unsigned square root of the 64-bit unsigned radicand.*
- static uint16\_t **MMAU\_s\_usqr\_1** (register uint32\_t lrad)
 

*Compute and return a 16-bit unsigned square root of the 32-bit unsigned radicand.*
- static uint32\_t **MMAU\_1\_usqra** (void)
 

*Compute and return a 32-bit unsigned square root of the radicand stored in the A10 register of the MMAU.*

## Signed Integer Operands

- static void **MMAU\_slda\_d** (register int64\_t dval)
 

*Load A10 accumulator register of the MMAU by 64-bit integer value.*
- static int64\_t **MMAU\_d\_smul\_ll** (register int32\_t lval1, register int32\_t lval2)
 

*Multiply two 32-bit integer values returning a 64-bit integer product.*
- static int64\_t **MMAU\_d\_smul\_dl** (register int64\_t dval, register int32\_t lval)
 

*Multiply 64-bit integer value with 32-bit integer value returning a 64-bit integer product.*
- static int64\_t **MMAU\_d\_smuls\_dl** (register int64\_t dval, register int32\_t lval)
 

*Saturating multiply 64-bit integer value with 32-bit integer value returning saturated 64-bit integer product.*
- static int64\_t **MMAU\_d\_smula\_1** (register int32\_t lval)
 

*Multiply 32-bit integer value with 64-bit integer value stored in the A10 register of the MMAU returning a 64-bit integer product.*
- static int64\_t **MMAU\_d\_smulas\_1** (register int32\_t lval)

Saturating multiply 32-bit integer value with 64-bit integer value stored in the A10 register of the MMAU returning saturated 64-bit integer product.

- static int64\_t **MMAU\_d\_smac\_ll** (register int32\_t lval1, register int32\_t lval2)
 

*Multiply two 32-bit integer values and add product with value stored in the A10 register of the MMAU returning a 64-bit integer A10 register value.*
- static int64\_t **MMAU\_d\_smaes\_ll** (register int32\_t lval1, register int32\_t lval2)
 

*Saturating multiply two 32-bit integer values and add product with value stored in the A10 register of the MMAU returning a 64-bit integer A10 register value.*
- static int64\_t **MMAU\_d\_smac\_dl** (register int64\_t dval, register int32\_t lval)
 

*Multiply 64-bit integer value with 32-bit integer value and add product with value stored in the A10 register of the MMAU returning a 64-bit integer A10 register value.*
- static int64\_t **MMAU\_d\_smacs\_dl** (register int64\_t dval, register int32\_t lval)
 

*Saturating multiply 64-bit integer value with 32-bit integer value and add product with value stored in the A10 register of the MMAU returning saturated 64-bit integer A10 register value.*
- static int64\_t **MMAU\_d\_smaca\_dl** (register int64\_t dval, register int32\_t lval)
 

*Multiply 32-bit integer value by value stored in the A10 register of the MMAU and add product with 64-bit integer value returning a 64-bit integer A10 register value.*
- static int64\_t **MMAU\_d\_smacas\_dl** (register int64\_t dval, register int32\_t lval)
 

*Saturating multiply 32-bit integer value by value stored in the A10 register of the MMAU and add product with 64-bit integer value returning a saturated 64-bit integer A10 register value.*
- static int32\_t **MMAU\_l\_sdiv\_ll** (register int32\_t lnum, register int32\_t lden)
 

*Divide two 32-bit integer values returning a 32-bit integer quotient.*
- static int32\_t **MMAU\_l\_sdivs\_ll** (register int32\_t lnum, register int32\_t lden)
 

*Divide two 32-bit integer values returning a 32-bit integer quotient.*
- static int64\_t **MMAU\_d\_sdiv\_dl** (register int64\_t dnum, register int32\_t lden)
 

*Divide 64-bit integer value by 32-bit integer value returning a 64-bit integer quotient.*
- static int64\_t **MMAU\_d\_sdivs\_dl** (register int64\_t dnum, register int32\_t lden)
 

*Divide 64-bit integer value by 32-bit integer value returning a 64-bit integer quotient.*
- static int64\_t **MMAU\_d\_sdiv\_dd** (register int64\_t dnum, register int64\_t dden)
 

*Divide two 64-bit integer values returning a 64-bit integer quotient.*
- static int64\_t **MMAU\_d\_sdivs\_dd** (register int64\_t dnum, register int64\_t dden)
 

*Divide two 64-bit integer values returning a 64-bit integer quotient.*
- static int64\_t **MMAU\_d\_sdiva\_l** (register int32\_t lden1)
 

*Divide 32-bit integer value stored in the A10 register of the MMAU by 32-bit integer value returning a 64-bit integer quotient.*
- static int64\_t **MMAU\_d\_sdivas\_l** (register int32\_t lden1)
 

*Divide 32-bit integer value stored in the A10 register of the MMAU by 32-bit integer value returning saturated 64-bit integer quotient.*
- static int64\_t **MMAU\_d\_sdiva\_d** (register int64\_t dden1)
 

*Divide 64-bit integer value stored in the A10 register of the MMAU by 64-bit integer value returning a 64-bit integer quotient.*
- static int64\_t **MMAU\_d\_sdivas\_d** (register int64\_t dden1)
 

*Divide 64-bit integer value stored in the A10 register of the MMAU by 64-bit integer value returning saturated 64-bit integer quotient.*

## Fractional Operands

- static void **MMAU\_lda\_d** (register **frac64\_t** dval)
 

*Load A10 accumulator register of the MMAU by 64-bit fractional value.*
- static **frac32\_t** **MMAU\_l\_mul\_ll** (register **frac32\_t** lval1, register **frac32\_t** lval2)

- static **frac32\_t MMAU\_1\_muls\_ll** (register **frac32\_t** lval1, register **frac32\_t** lval2)
 

*Multiply two 32-bit fractional values returning a 32-bit fractional product.*
- static **frac64\_t MMAU\_d\_mul\_ll** (register **frac32\_t** lval1, register **frac32\_t** lval2)
 

*Saturating multiply two 32-bit fractional values returning saturated 32-bit fractional product.*
- static **frac64\_t MMAU\_d\_muls\_ll** (register **frac32\_t** lval1, register **frac32\_t** lval2)
 

*Multiply two 32-bit fractional values returning a 64-bit fractional product.*
- static **frac64\_t MMAU\_d\_mulas\_ll** (register **frac32\_t** lval1, register **frac32\_t** lval2)
 

*Saturating multiply two 32-bit fractional values returning saturated 64-bit fractional product.*
- static **frac64\_t MMAU\_d\_mul\_dl** (register **frac64\_t** dval, register **frac32\_t** lval)
 

*Multiply 64-bit fractional value with 32-bit fractional value returning a 64-bit fractional product.*
- static **frac64\_t MMAU\_d\_muls\_dl** (register **frac64\_t** dval, register **frac32\_t** lval)
 

*Saturating multiply 64-bit fractional value with 32-bit fractional value returning saturated 64-bit fractional product.*
- static **frac64\_t MMAU\_d\_mula\_1** (register **frac32\_t** lval)
 

*Multiply 32-bit fractional value with 64-bit fractional value stored in the A10 register of the MMAU returning a 64-bit fractional product.*
- static **frac64\_t MMAU\_d\_mulas\_1** (register **frac32\_t** lval)
 

*Saturating multiply 32-bit fractional value with 64-bit fractional value stored in the A10 register of the MMAU returning saturated 64-bit fractional product.*
- static **frac32\_t MMAU\_1\_mul\_dl** (register **frac64\_t** dval, register **frac32\_t** lval)
 

*Multiply 64-bit fractional value with 32-bit fractional value returning a 32-bit fractional product.*
- static **frac32\_t MMAU\_1\_muls\_dl** (register **frac64\_t** dval, register **frac32\_t** lval)
 

*Saturating multiply 64-bit fractional value with 32-bit fractional value returning saturated 32-bit fractional product.*
- static **frac32\_t MMAU\_1\_mula\_1** (register **frac32\_t** lval)
 

*Multiply 32-bit fractional value with 64-bit fractional value stored in the A10 register of the MMAU returning a 32-bit fractional product.*
- static **frac32\_t MMAU\_1\_mulas\_1** (register **frac32\_t** lval)
 

*Saturating multiply 32-bit fractional value with 64-bit fractional value stored in the A10 register of the MMAU returning saturated 32-bit fractional product.*
- static **frac64\_t MMAU\_d\_mac\_ll** (register **frac32\_t** lval1, register **frac32\_t** lval2)
 

*Multiply two 32-bit fractional values and add product with value stored in the A10 register of the MMAU returning a 64-bit fractional A10 register value.*
- static **frac64\_t MMAU\_d\_macs\_ll** (register **frac32\_t** lval1, register **frac32\_t** lval2)
 

*Saturating multiply two 32-bit fractional values and add product with value stored in the A10 register of the MMAU returning a 64-bit fractional A10 register value.*
- static **frac64\_t MMAU\_d\_mac\_dl** (register **frac64\_t** dval, register **frac32\_t** lval)
 

*Multiply 64-bit fractional value with 32-bit fractional value and add product with value stored in the A10 register of the MMAU returning a 64-bit fractional A10 register value.*
- static **frac64\_t MMAU\_d\_macs\_dl** (register **frac64\_t** dval, register **frac32\_t** lval)
 

*Saturating multiply 64-bit fractional value with 32-bit fractional value and add product with value stored in the A10 register of the MMAU returning saturated 64-bit fractional A10 register value.*
- static **frac64\_t MMAU\_d\_maca\_dl** (register **frac64\_t** dval, register **frac32\_t** lval)
 

*Multiply 32-bit fractional value by value stored in the A10 register of the MMAU and add product with 64-bit fractional value returning a 64-bit fractional A10 register value.*
- static **frac64\_t MMAU\_d\_macas\_dl** (register **frac64\_t** dval, register **frac32\_t** lval)
 

*Saturating multiply 32-bit fractional value by value stored in the A10 register of the MMAU and add product with 64-bit fractional value returning a saturated 64-bit fractional A10 register value.*
- static **frac32\_t MMAU\_1\_mac\_ll** (register **frac32\_t** lval1, register **frac32\_t** lval2)
 

*Multiply two 32-bit fractional values and add product with value stored in the A10 register of the MMAU returning a 32-bit fractional A10 register value.*

- static `frac32_t MMAU_1_macs_ll` (register `frac32_t lval1`, register `frac32_t lval2`)  
*Saturating multiply two 32-bit fractional values and add product with value stored in the A10 register of the MMAU returning a 32-bit fractional A10 register value.*
- static `frac32_t MMAU_1_mac_dl` (register `frac64_t dval`, register `frac32_t lval`)  
*Multiply 64-bit fractional value with 32-bit fractional value and add product with value stored in the A10 register of the MMAU returning a 32-bit fractional A10 register value.*
- static `frac32_t MMAU_1_macs_dl` (register `frac64_t dval`, register `frac32_t lval`)  
*Saturating multiply 64-bit fractional value with 32-bit fractional value and add product with value stored in the A10 register of the MMAU returning saturated 32-bit fractional A10 register value.*
- static `frac32_t MMAU_1_maca_dl` (register `frac64_t dval`, register `frac32_t lval`)  
*Multiply 32-bit fractional value by value stored in the A10 register of the MMAU and add product with 64-bit fractional value returning a 32-bit fractional A10 register value.*
- static `frac32_t MMAU_1_macas_dl` (register `frac64_t dval`, register `frac32_t lval`)  
*Saturating multiply 32-bit fractional value by value stored in the A10 register of the MMAU and add product with 64-bit fractional value returning a saturated 32-bit fractional A10 register value.*
- static `frac32_t MMAU_1_div_ll` (register `frac32_t lnum`, register `frac32_t lden`)  
*Divide two 32-bit fractional values returning a 32-bit fractional quotient.*
- static `frac32_t MMAU_1_divs_ll` (register `frac32_t lnum`, register `frac32_t lden`)  
*Divide two 32-bit fractional values returning a 32-bit fractional quotient.*
- static `frac32_t MMAU_1_divas_1` (register `frac32_t lden`)  
*Divide 64-bit fractional value stored in the A10 register of the MMAU by 32-bit fractional value returning saturated 32-bit fractional quotient.*
- static `frac64_t MMAU_d_div_dl` (register `frac64_t dnum`, register `frac32_t lden`)  
*Divide 64-bit fractional value by 32-bit fractional value returning a 64-bit fractional quotient.*
- static `frac64_t MMAU_d_divs_dl` (register `frac64_t dnum`, register `frac32_t lden`)  
*Divide 64-bit fractional value by 32-bit fractional value returning a 64-bit fractional quotient.*
- static `frac64_t MMAU_d_diva_1` (register `frac32_t lden1`)  
*Divide 32-bit fractional value stored in the A10 register of the MMAU by 32-bit fractional value returning a 64-bit fractional quotient.*
- static `frac64_t MMAU_d_divas_1` (register `frac32_t lden1`)  
*Divide 32-bit fractional value stored in the A10 register of the MMAU by 32-bit fractional value returning saturated 64-bit fractional quotient.*
- static `frac32_t MMAU_1_diva_1` (register `frac32_t lden`)  
*Divide 64-bit fractional value stored in the A10 register of the MMAU by 32-bit fractional value returning a 32-bit fractional quotient.*
- static `frac32_t MMAU_1_sqr_1` (register `frac32_t lrad`)  
*Compute and return a 32-bit fractional square root of the 32-bit fractional radicand.*
- static `frac32_t MMAU_1_sqr_d` (register `frac64_t drad`)  
*Compute and return a 32-bit fractional square root of the 64-bit fractional radicand.*
- static `frac32_t MMAU_1_sqra` (void)  
*Compute and return a 32-bit fractional square root of the radicand stored in the A10 register of the MMAU.*

## 20.4 Enumeration Type Documentation

### 20.4.1 enum mmau\_interrupt\_enable\_t

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

Enumerator

***kMMAU\_AccumOverflowInterruptEnable*** Accumulation Overflow Enable.

***kMMAU\_OverflowInterruptEnable*** Multiply or Divide overflow Enable.

***kMMAU\_DivideByZeroInterruptEnable*** Divide by Zero Enable.

## 20.4.2 enum mmau\_interrupt\_flag\_t

Enumerator

***kMMAU\_AccumOverflowInterruptFlag*** Accumulation Overflow Interrupt Flag.

***kMMAU\_OverflowInterruptFlag*** Multiply or Divide overflow Interrupt Flag.

***kMMAU\_DivideByZeroInterruptFlag*** Divide by Zero Interrupt Flag.

## 20.4.3 enum mmau\_instruction\_flag\_t

Enumerator

***kMMAU\_AccumOverflowInstructionFlag*** Accumulation Overflow.

***kMMAU\_OverflowInstructionFlag*** Multiply or Divide overflow.

***kMMAU\_DivideByZeroInstructionFlag*** Divide by Zero.

***kMMAU\_NegativeInstructionFlag*** Signed calculation result is negative.

## 20.5 Function Documentation

### 20.5.1 static void MMAU\_EnableDMA ( MMAU\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	MMAU peripheral address.
<i>enable</i>	Mode of DMA access <ul style="list-style-type: none"> <li>• true Enable DMA access</li> <li>• false Disable DMA access</li> </ul>

### 20.5.2 static void MMAU\_EnableSupervisorOnly ( MMAU\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	MMAU peripheral address.
<i>enable</i>	Mode of MMAU module can be access <ul style="list-style-type: none"> <li>• true MMAU registers can only be access in Supervisor Mode.</li> <li>• false MMAU registers can be access in both User Mode or Supervisor Mode.</li> </ul>

### 20.5.3 void MMAU\_Reset( MMAU\_Type \* *base* )

This function sets control/status register to a known state. This state is defined in Reference Manual, which is power on reset value. This function must execute in a Supervisor Mode

Parameters

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

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

This function enables the interrupts related to the mask. Example:

```
MMAU_EnableInterrupts(MMAU,
    kMMAU_AccumOverflowInterruptEnable |
    kMMAU_DivideByZeroInterruptEnable);
```

Parameters

<i>base</i>	MMAU peripheral address.
<i>mask</i>	Mask of the interrupt enable to be written (kMMAU_AccumOverflowInterruptEnable kMMAU_OverflowInterruptEnable kMMAU_DivideByZeroInterruptEnable).

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

This function disables the interrupt related to the mask. Example:

```
MMAU_DisableInterrupts(MMAU,
    kMMAU_AccumOverflowInterruptEnable |
    kMMAU_DivideByZeroInterruptEnable);
```

Parameters

<i>base</i>	MMAU peripheral address.
<i>mask</i>	Mask of the interrupt enable to be written (kMMAU_AccumOverflowInterruptEnable kMMAU_OverflowInterruptEnable kMMAU_DivideByZeroInterruptEnable).

### 20.5.6 static uint32\_t MMAU\_GetEnabledInterrupts ( MMAU\_Type \* *base* ) [inline], [static]

This function gets all interrupt values. Example:

```
uint32_t flags = 0;
...
flags = MMAU_GetEnabledInterrupts(MMAU);
if (flags & kMMAU_AccumOverflowInterruptEnable)
{
...
}
```

Parameters

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

Returns

Combination of enabled interrupt

### 20.5.7 static uint32\_t MMAU\_GetInterruptFlags ( MMAU\_Type \* *base* ) [inline], [static]

This function gets interrupt flags.

Parameters

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

Returns

the mask of these interrupt flag bits.

### 20.5.8 void MMAU\_ClearInterruptFlags ( MMAU\_Type \* *base*, uint32\_t *mask* )

This function clears the interrupt flags. Example, if you want to clear Overflow and DivideByZero interrupt flags:

```
MMAU_ClearInterruptFlags(MMAU,
    kMMAU_OverflowInterruptFlag |
    kMMAU_DivideByZeroInterruptFlag);
```

Parameters

<i>base</i>	MMAU peripheral address.
<i>mask</i>	Mask of the asserted interrupt flags (kMMAU_AccumOverflowInterruptFlag kMMAU_OverflowInterruptFlag kMMAU_DivideByZeroInterruptFlag).

### 20.5.9 static uint32\_t MMAU\_GetInstructionFlags ( MMAU\_Type \* *base* ) [inline], [static]

This function gets the instruction flag. Instruction flags are updated by the MMAU after computation of each instruction. Example:

```
uint32_t flags;
...
flags = MMAU_GetInstructionFlags(MMAU);
if (flags & kMMAU_OverflowInstructionFlag)
{
    ...
}
```

Parameters

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

Returns

Combination of all instruction flags.

### 20.5.10 void MMAU\_SetInstructionFlags ( MMAU\_Type \* *base*, uint32\_t *mask* )

This function sets the instruction flags. Example:

```
MMAU_SetInstructionFlags(MMAU,
    kMMAU_AccumOverflowInstructionFlag |
    kMMAU_NegativeInstructionFlag);
MMAU_SetInstructionFlags(MMAU, kMMAU_OverflowInstructionFlag |
    kMMAU_DivideByZeroInstructionFlag);
```

Parameters

<i>base</i>	MMAU peripheral address.
<i>mask</i>	Mask of the instruction flags to be written

(kMMAU\_AccumOverflowInstructionFlag|kMMAU\_OverflowInstructionFlag|kMMAU\_DivideByZeroInstructionFlag|kMMAU\_NegativeInstructionFlag).

### 20.5.11 void MMAU\_ClearInstructionFlags ( MMAU\_Type \* *base*, uint32\_t *mask* )

This function clears the instruction flags. Example, if you want to clear Overflow and DivideByZero instruction flags:

```
MMAU_ClearInstructionFlags(MMAU, kMMAU_OverflowInstructionFlag|
                           kMMAU_DivideByZeroInstructionFlag);
```

Parameters

<i>base</i>	MMAU peripheral address.
<i>mask</i>	Mask of the asserted instruction flags

(kMMAU\_AccumOverflowInstructionFlag|kMMAU\_OverflowInstructionFlag|kMMAU\_DivideByZeroInstructionFlag|kMMAU\_NegativeInstructionFlag).

### 20.5.12 static uint32\_t MMAU\_GetHwRevCmd ( MMAU\_Type \* *base* ) [inline], [static]

This function gets the hardware revision level of the MMAU. It returns HDR field of the control/status register.

Parameters

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

Returns

uint32\_t hardware revision level.

### 20.5.13 static void MMAU\_udal\_d ( register uint64\_t *dval* ) [inline], [static]

The ulda\_d function loads A10 accumulator register of the MMAU by 64-bit unsigned value.

Parameters

<i>dval</i>	uint64_t unsigned load value.
-------------	-------------------------------

#### 20.5.14 static uint64\_t MMAU\_d\_umul\_ll ( register uint32\_t *lval1*, register uint32\_t *lval2* ) [inline], [static]

The [MMAU\\_d\\_umul\\_ll](#) function multiplies two 32-bit unsigned values returning a 64-bit unsigned product.

Parameters

<i>lval1</i>	uint32_t unsigned value.
<i>lval2</i>	uint32_t unsigned value.

Returns

uint64\_t unsigned value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.15 static uint64\_t MMAU\_d\_umul\_dl ( register uint64\_t *dval*, register uint32\_t *lval* ) [inline], [static]

The [MMAU\\_d\\_umul\\_dl](#) function multiplies 64-bit unsigned value with 32-bit unsigned value returning a 64-bit unsigned product.

Parameters

<i>dval</i>	uint64_t unsigned value.
<i>lval</i>	uint32_t unsigned value.

Returns

uint64\_t unsigned value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

**20.5.16 static uint64\_t MMAU\_d\_umuls\_dl ( register uint64\_t *dval*, register uint32\_t *ival* ) [inline], [static]**

The [MMAU\\_d\\_umuls\\_dl](#) function multiplies 64-bit unsigned value with 32-bit unsigned value returning saturated 64-bit unsigned product.

Parameters

<i>dval</i>	uint64_t unsigned value.
<i>lval</i>	uint32_t unsigned value.

Returns

uint64\_t unsigned value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation.

#### 20.5.17 static uint64\_t MMAU\_d\_umula\_l( register uint32\_t *lval* ) [inline], [static]

The [MMAU\\_d\\_umula\\_l](#) function multiplies 32-bit unsigned value with 64-bit unsigned value stored in the A10 register of the MMAU returning a 64-bit unsigned product.

Parameters

<i>lval</i>	uint32_t unsigned value.
-------------	--------------------------

Returns

uint64\_t unsigned value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.18 static uint64\_t MMAU\_d\_umulas\_l( register uint32\_t *lval* ) [inline], [static]

The [MMAU\\_d\\_umulas\\_l](#) function multiplies 32-bit unsigned value with 64-bit unsigned value stored in the A10 register of the MMAU returning saturated 64-bit unsigned product.

Parameters

<i>lval</i>	uint32_t unsigned value.
-------------	--------------------------

Returns

uint64\_t unsigned value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation.

#### 20.5.19 static uint64\_t MMAU\_d\_umac\_ll ( register uint32\_t *lval1*, register uint32\_t *lval2* ) [inline], [static]

The [MMAU\\_d\\_umac\\_ll](#) function multiplies two 32-bit unsigned values and add product with value stored in the A10 register of the MMAU returning a 64-bit unsigned A10 register value.

Parameters

<i>lval1</i>	uint32_t unsigned value.
<i>lval2</i>	uint32_t unsigned value.

Returns

uint64\_t unsigned value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.20 static uint64\_t MMAU\_d\_umacs\_ll ( register uint32\_t *lval1*, register uint32\_t *lval2* ) [inline], [static]

The [MMAU\\_d\\_umacs\\_ll](#) function multiplies two 32-bit unsigned values and add product with value stored in the A10 register of the MMAU returning saturated 64-bit unsigned A10 register value.

Parameters

<i>lval1</i>	uint32_t unsigned value.
<i>lval2</i>	uint32_t unsigned value.

Returns

uint64\_t unsigned value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation.

#### 20.5.21 static uint64\_t MMAU\_d\_umac\_dl ( register uint64\_t *dval*, register uint32\_t *lval* ) [inline], [static]

The [MMAU\\_d\\_umac\\_dl](#) function multiplies 64-bit unsigned value with 32-bit unsigned value and add product with value stored in the A10 register of the MMAU returning a 64-bit unsigned A10 register value.

Parameters

<i>dval</i>	uint64_t unsigned value.
<i>lval</i>	uint32_t unsigned value.

Returns

uint64\_t unsigned value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.22 static uint64\_t MMAU\_d\_umacs\_dl ( register uint64\_t *dval*, register uint32\_t *lval* ) [inline], [static]

The [MMAU\\_d\\_umacs\\_dl](#) function multiplies 64-bit unsigned value with 32-bit unsigned value and add product with value stored in the A10 register of the MMAU returning saturated 64-bit unsigned A10 register value.

Parameters

<i>dval</i>	uint64_t unsigned value.
<i>lval</i>	uint32_t unsigned value.

Returns

uint64\_t unsigned value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation.

#### 20.5.23 static uint64\_t MMAU\_d\_umaca\_dl ( register uint64\_t *dval*, register uint32\_t *lval* ) [inline], [static]

The [MMAU\\_d\\_umaca\\_dl](#) function multiplies 32-bit unsigned value by value stored in the A10 register of the MMAU and add product with 64-bit unsigned value returning a 64-bit unsigned A10 register value.

Parameters

<i>dval</i>	uint64_t unsigned value.
<i>lval</i>	uint32_t unsigned value.

Returns

uint64\_t unsigned value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.24 static uint64\_t MMAU\_d\_umacas\_dl ( register uint64\_t *dval*, register uint32\_t *lval* ) [inline], [static]

The [MMAU\\_d\\_umacas\\_dl](#) function multiplies 32-bit unsigned value by value stored in the A10 register of the MMAU and add product with 64-bit unsigned value returning saturated 64-bit unsigned A10 register value.

Parameters

<i>dval</i>	uint64_t unsigned value.
<i>lval</i>	uint32_t unsigned value.

Returns

uint64\_t unsigned value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation.

#### 20.5.25 static uint32\_t MMAU\_l\_udiv\_ll ( register uint32\_t *lnum*, register uint32\_t *lden* ) [inline], [static]

The [MMAU\\_l\\_udiv\\_ll](#) function divides two 32-bit unsigned values returning a 32-bit unsigned quotient.

Parameters

<i>lnum</i>	uint32_t unsigned divisor value.
<i>lden</i>	uint32_t unsigned dividend value.

Returns

uint32\_t unsigned quotient value.

Note

Quotient is stored in A0 register of the MMAU for next computation.

#### 20.5.26 static uint64\_t MMAU\_d\_udiv\_dl ( register uint64\_t *dnum*, register uint32\_t *lden* ) [inline], [static]

The [MMAU\\_d\\_udiv\\_dl](#) function divides 64-bit unsigned value by 32-bit unsigned value returning a 64-bit unsigned quotient.

Parameters

<i>dnum</i>	uint64_t unsigned divisor value.
<i>lden</i>	uint32_t unsigned dividend value.

Returns

uint64\_t unsigned quotient value.

Note

Quotient is stored in A10 register of the MMAU for next computation.

#### 20.5.27 static uint64\_t MMAU\_d\_udiv\_dd ( register uint64\_t *dnum*, register uint64\_t *dden* ) [inline], [static]

The [MMAU\\_d\\_udiv\\_dd](#) function divides two 64-bit unsigned values returning a 64-bit unsigned quotient.

Parameters

<i>dnum</i>	uint64_t unsigned divisor value.
<i>dden</i>	uint64_t unsigned dividend value.

Returns

uint64\_t unsigned quotient value.

Note

Quotient is stored in A10 register of the MMAU for next computation.

#### 20.5.28 static uint64\_t MMAU\_d\_udiva\_l ( register uint32\_t *lden1* ) [inline], [static]

The [MMAU\\_d\\_udiva\\_l](#) function divides 64-bit unsigned value stored in the A10 register of the MMAU by 32-bit unsigned value returning a 64-bit unsigned quotient.

Parameters

<i>lden1</i>	uint32_t unsigned dividend value.
--------------	-----------------------------------

Returns

uint64\_t unsigned quotient value.

Note

Quotient is stored in A10 register of the MMAU for next computation.

#### 20.5.29 static uint64\_t MMAU\_d\_udiva\_d ( register uint64\_t *dden1* ) [inline], [static]

The [MMAU\\_d\\_udiva\\_d](#) function divides 64-bit unsigned value stored in the A10 register of the MMAU by 64-bit unsigned value returning a 64-bit unsigned quotient.

Parameters

<i>dden1</i>	uint64_t unsigned dividend value.
--------------	-----------------------------------

Returns

uint64\_t unsigned quotient value.

Note

Quotient is stored in A10 register of the MMAU for next computation.

#### 20.5.30 static uint32\_t MMAU\_l\_usqr\_l ( register uint32\_t *lrad* ) [inline], [static]

The [MMAU\\_l\\_usqr\\_l](#) function computes and returns a 32-bit unsigned square root of the 32-bit unsigned radicand.

Parameters

<i>lrad</i>	uint32_t unsigned radicand.
-------------	-----------------------------

Returns

uint32\_t unsigned square root.

Note

Quotient is stored in A0 register of the MMAU for next computation.

#### 20.5.31 static uint32\_t MMAU\_l\_usqr\_d ( register uint64\_t *drad* ) [inline], [static]

The [MMAU\\_l\\_usqr\\_d](#) function computes and returns a 32-bit unsigned square root of the 64-bit unsigned radicand.

Parameters

<i>drad</i>	uint64_t unsigned radicand.
-------------	-----------------------------

Returns

uint32\_t unsigned square root.

Note

Quotient is stored in A0 register of the MMAU for next computation.

#### 20.5.32 static uint16\_t MMAU\_s\_usqr\_l ( register uint32\_t *lrad* ) [inline], [static]

The [MMAU\\_s\\_usqr\\_l](#) function computes and returns a 16-bit unsigned square root of the 32-bit unsigned radicand.

Parameters

<i>lrad</i>	uint32 unsigned radicand.
-------------	---------------------------

Returns

uint16 unsigned square root.

Note

Square root is stored in A0 register of the MMAU for next computation.

### 20.5.33 static uint32\_t MMAU\_l\_usqra( void ) [inline], [static]

The [MMAU\\_l\\_usqra](#) function computes and returns a 32-bit unsigned square root of the radicand stored in the A10 register of the MMAU.

Returns

uint32\_t unsigned square root.

Note

Quotient is stored in A0 register of the MMAU for next computation.

### 20.5.34 static void MMAU\_slda\_d( register int64\_t *dval* ) [inline], [static]

The [MMAU\\_slda\\_d](#) function loads A10 accumulator register of the MMAU by 64-bit integer value.

Parameters

<i>dval</i>	int64_t integer value.
-------------	------------------------

### 20.5.35 static int64\_t MMAU\_d\_smul\_ll( register int32\_t *lval1*, register int32\_t *lval2* ) [inline], [static]

The [MMAU\\_d\\_smul\\_ll](#) function multiplies two 32-bit integer values returning a 64-bit integer product.

Parameters

<i>lval1</i>	int32_t integer value.
<i>lval2</i>	int32_t integer value.

Returns

int64\_t integer value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.36 static int64\_t MMAU\_d\_smul\_dl ( register int64\_t *dval*, register int32\_t *lval* ) [inline], [static]

The [MMAU\\_d\\_smul\\_dl](#) function multiplies 64-bit integer value with 32-bit integer value returning a 64-bit integer product.

Parameters

<i>dval</i>	int64_t integer value.
<i>lval</i>	int32_t integer value.

Returns

int64\_t integer value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.37 static int64\_t MMAU\_d\_smuls\_dl ( register int64\_t *dval*, register int32\_t *lval* ) [inline], [static]

The [MMAU\\_d\\_smuls\\_dl](#) function multiplies 64-bit integer value with 32-bit integer value returning saturated 64-bit integer product.

Parameters

<i>dval</i>	int64_t integer value.
<i>lval</i>	int32_t integer value.

Returns

int64\_t integer value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation.

#### 20.5.38 static int64\_t MMAU\_d\_smula\_l( register int32\_t *lval* ) [inline], [static]

The [MMAU\\_d\\_smula\\_l](#) function multiplies 32-bit integer value with 64-bit integer value stored in the A10 register of the MMAU returning a 64-bit integer product.

Parameters

<i>lval</i>	int32_t integer value.
-------------	------------------------

Returns

int64\_t integer value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.39 static int64\_t MMAU\_d\_smulas\_l( register int32\_t *lval* ) [inline], [static]

The [MMAU\\_d\\_smulas\\_l](#) function multiplies 32-bit integer value with 64-bit integer value stored in the A10 register of the MMAU returning saturated 64-bit integer product.

Parameters

<i>lval</i>	int32_t integer value.
-------------	------------------------

Returns

int64\_t integer value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation.

#### 20.5.40 static int64\_t MMAU\_d\_smac\_ll ( register int32\_t *lval1*, register int32\_t *lval2* ) [inline], [static]

The [MMAU\\_d\\_smac\\_ll](#) function multiplies two 32-bit integer values and add product with value stored in the A10 register of the MMAU returning a 64-bit integer A10 register value.

Parameters

<i>lval1</i>	int32_t integer value.
<i>lval2</i>	int32_t integer value.

Returns

int64\_t integer value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.41 static int64\_t MMAU\_d\_smacs\_ll ( register int32\_t *lval1*, register int32\_t *lval2* ) [inline], [static]

The [MMAU\\_d\\_smacs\\_ll](#) function multiplies two 32-bit integer values and add product with value stored in the A10 register of the MMAU returning saturated 64-bit integer A10 register value.

Parameters

<i>lval1</i>	int32_t integer value.
<i>lval2</i>	int32_t integer value.

Returns

int64\_t integer value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation.

#### 20.5.42 static int64\_t MMAU\_d\_smac\_dl ( register int64\_t *dval*, register int32\_t *lval* ) [inline], [static]

The [MMAU\\_d\\_smac\\_dl](#) function multiplies 64-bit integer value with 32-bit integer value and add product with value stored in the A10 register of the MMAU returning a 64-bit integer A10 register value.

Parameters

<i>dval</i>	int64_t integer value.
<i>lval</i>	int32_t integer value.

Returns

int64\_t integer value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.43 static int64\_t MMAU\_d\_smacs\_dl ( register int64\_t *dval*, register int32\_t *lval* ) [inline], [static]

The [MMAU\\_d\\_smacs\\_dl](#) function multiplies 64-bit integer value with 32-bit integer value and add product with value stored in the A10 register of the MMAU returning saturated 64-bit integer A10 register value.

Parameters

<i>dval</i>	int64_t integer value.
<i>lval</i>	int32_t integer value.

Returns

int64\_t integer value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation. If saturation occurs, the instruction sets the accumulation overflow (Q) and multiply or divide overflow (V) flags to 1 in the CSR. Otherwise, it clears the Q and V flags.

#### 20.5.44 static int64\_t MMAU\_d\_smaca\_dl ( register int64\_t *dval*, register int32\_t *lval* ) [inline], [static]

The [MMAU\\_d\\_smaca\\_dl](#) function multiplies 32-bit integer value by value stored in the A10 register of the MMAU and add product with 64-bit integer value returning a 64-bit integer A10 register value.

Parameters

<i>dval</i>	int64_t integer value.
<i>lval</i>	int32_t integer value.

Returns

int64\_t integer value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.45 static int64\_t MMAU\_d\_smacas\_dl ( register int64\_t *dval*, register int32\_t *lval* ) [inline], [static]

The [MMAU\\_d\\_smacas\\_dl](#) function multiplies 32-bit integer value by value stored in the A10 register of the MMAU and add product with 64-bit integer value returning saturated 64-bit integer A10 register value.

Parameters

<i>dval</i>	int64_t integer value.
<i>lval</i>	int32_t integer value.

Returns

int64\_t integer value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation. If saturation occurs, the instruction sets the accumulation overflow (Q) and multiply or divide overflow (V) flags to 1 in the CSR. Otherwise, it clears the Q and V flags.

#### 20.5.46 static int32\_t MMAU\_I\_sdiv\_ll( register int32\_t *Inum*, register int32\_t *Iden* ) [inline], [static]

The [MMAU\\_I\\_sdiv\\_ll](#) function divides two 32-bit integer values returning a 32-bit integer quotient.

Parameters

<i>Inum</i>	int32_t integer divisor value.
<i>Iden</i>	int32_t integer dividend value.

Returns

int32\_t integer quotient value.

Note

Quotient is stored in A0 register of the MMAU for next computation.

#### 20.5.47 static int32\_t MMAU\_I\_sdivs\_ll( register int32\_t *Inum*, register int32\_t *Iden* ) [inline], [static]

The [MMAU\\_I\\_sdivs\\_ll](#) function divides two 32-bit integer values returning a 32-bit integer quotient.

Parameters

<i>lnum</i>	int32_t integer divisor value.
<i>lden</i>	int32_t integer dividend value.

Returns

int32\_t integer quotient value.

Note

Saturated quotient is stored in A0 register of the MMAU for next computation.

#### 20.5.48 static int64\_t MMAU\_d\_sdiv\_dl ( register int64\_t *dnum*, register int32\_t *lden* ) [inline], [static]

The [MMAU\\_d\\_sdiv\\_dl](#) function divides 64-bit integer value by 32-bit integer value returning a 64-bit integer quotient.

Parameters

<i>dnum</i>	int64_t integer divisor value.
<i>lden</i>	int32_t integer dividend value.

Returns

int64\_t integer quotient value.

Note

Quotient is stored in A10 register of the MMAU for next computation.

#### 20.5.49 static int64\_t MMAU\_d\_sdivs\_dl ( register int64\_t *dnum*, register int32\_t *lden* ) [inline], [static]

The [MMAU\\_d\\_sdivs\\_dl](#) function divides 64-bit integer value by 32-bit integer value returning a 64-bit integer quotient.

Parameters

<i>dnum</i>	int64_t integer divisor value.
<i>lden</i>	int32_t integer dividend value.

Returns

int64\_t integer quotient value.

Note

Saturated quotient is stored in A10 register of the MMAU for next computation.

#### 20.5.50 static int64\_t MMAU\_d\_sdiv\_dd ( register int64\_t *dnum*, register int64\_t *dden* ) [inline], [static]

The [MMAU\\_d\\_sdiv\\_dd](#) function divides two 64-bit integer values returning a 64-bit integer quotient.

Parameters

<i>dnum</i>	int64_t integer divisor value.
<i>dden</i>	int64_t integer dividend value.

Returns

int64\_t integer quotient value.

Note

Quotient is stored in A10 register of the MMAU for next computation.

#### 20.5.51 static int64\_t MMAU\_d\_sdivs\_dd ( register int64\_t *dnum*, register int64\_t *dden* ) [inline], [static]

The [MMAU\\_d\\_sdivs\\_dd](#) function divides two 64-bit integer values returning a 64-bit integer quotient.

Parameters

<i>dnum</i>	int64_t integer divisor value.
<i>dden</i>	int64_t integer dividend value.

Returns

int64\_t integer quotient value.

Note

Saturated quotient is stored in A10 register of the MMAU for next computation.

#### 20.5.52 static int64\_t MMAU\_d\_sdiva\_I ( register int32\_t *lden1* ) [inline], [static]

The [MMAU\\_d\\_sdiva\\_I](#) function divides 32-bit integer value stored in the A10 register of the MMAU by 32-bit integer value returning a 64-bit integer quotient.

Parameters

<i>lden1</i>	int32_t integer dividend value.
--------------	---------------------------------

Returns

int64\_t integer quotient value.

Note

Quotient is stored in A10 register of the MMAU for next computation.

#### 20.5.53 static int64\_t MMAU\_d\_sdivas\_I ( register int32\_t *lden1* ) [inline], [static]

The [MMAU\\_d\\_sdivas\\_I](#) function divides 32-bit integer value stored in the A10 register of the MMAU by 32-bit integer value returning a saturated 64-bit integer quotient.

Parameters

<i>lden1</i>	int32_t integer dividend value.
--------------	---------------------------------

Returns

int64\_t integer quotient value.

Note

Saturated quotient is stored in A10 register of the MMAU for next computation.

#### 20.5.54 static int64\_t MMAU\_d\_sdiva\_d ( register int64\_t *dden1* ) [inline], [static]

The [MMAU\\_d\\_sdiva\\_d](#) function divides 64-bit integer value stored in the A10 register of the MMAU by 64-bit integer value returning a 64-bit integer quotient.

Parameters

<i>dden1</i>	int64_t integer dividend value.
--------------	---------------------------------

Returns

int64\_t integer quotient value.

Note

Quotient is stored in A10 register of the MMAU for next computation.

#### 20.5.55 static int64\_t MMAU\_d\_sdivas\_d ( register int64\_t *dden1* ) [inline], [static]

The [MMAU\\_d\\_sdivas\\_d](#) function divides 64-bit integer value stored in the A10 register of the MMAU by 64-bit integer value returning a saturated 64-bit integer quotient.

Parameters

<i>dden1</i>	int64_t integer dividend value.
--------------	---------------------------------

Returns

int64\_t integer quotient value.

Note

Saturated quotient is stored in A10 register of the MMAU for next computation.

### 20.5.56 static void MMAU\_lda\_d ( register frac64\_t *dval* ) [inline], [static]

The [MMAU\\_lda\\_d](#) function loads A10 accumulator register of the MMAU by 64-bit fractional value.

Parameters

<i>dval</i>	frac64_t fractional value.
-------------	----------------------------

### 20.5.57 static frac32\_t MMAU\_l\_mul\_ll ( register frac32\_t *lval1*, register frac32\_t *lval2* ) [inline], [static]

The [MMAU\\_l\\_mul\\_ll](#) function multiplies two 32-bit fractional values returning a 32-bit fractional product.

Parameters

<i>lval1</i>	frac32_t fractional value.
<i>lval2</i>	frac32_t fractional value.

Returns

frac32\_t fractional value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

**20.5.58 static frac32\_t MMAU\_I\_muls\_ll ( register frac32\_t *lval1*, register frac32\_t *lval2* ) [inline], [static]**

The [MMAU\\_I\\_muls\\_ll](#) function multiplies two 32-bit fractional values returning saturated 32-bit fractional product.

Parameters

<i>lval1</i>	<a href="#">frac32_t</a> fractional value.
<i>lval2</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac32\\_t](#) fractional value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation.

#### 20.5.59 static [frac64\\_t](#) MMAU\_d\_mul\_ll ( [register frac32\\_t lval1](#), [register frac32\\_t lval2](#) ) [[inline](#)], [[static](#)]

The [MMAU\\_d\\_mul\\_ll](#) function multiplies two 32-bit fractional values returning a 64-bit fractional product.

Parameters

<i>lval1</i>	<a href="#">frac32_t</a> fractional value.
<i>lval2</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac64\\_t](#) fractional value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.60 static [frac64\\_t](#) MMAU\_d\_muls\_ll ( [register frac32\\_t lval1](#), [register frac32\\_t lval2](#) ) [[inline](#)], [[static](#)]

The [MMAU\\_d\\_muls\\_ll](#) function multiplies two 32-bit fractional values returning saturated 64-bit fractional product.

Parameters

<i>lval1</i>	<a href="#">frac32_t</a> fractional value.
<i>lval2</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac64\\_t](#) fractional value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation. If saturation occurs, the instruction sets the accumulation overflow (Q) and multiply or divide overflow (V) flags to 1 in the CSR. Otherwise, it clears the Q and V flags.

#### 20.5.61 static [frac64\\_t](#) MMAU\_d\_mul\_dl ( [register frac64\\_t dval](#), [register frac32\\_t lval](#) ) [[inline](#)], [[static](#)]

The [MMAU\\_d\\_mul\\_dl](#) function multiplies 64-bit fractional value with 32-bit fractional value returning a 64-bit fractional product.

Parameters

<i>dval</i>	<a href="#">frac64_t</a> fractional value.
<i>lval</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac64\\_t](#) fractional value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.62 static [frac64\\_t](#) MMAU\_d\_muls\_dl ( [register frac64\\_t dval](#), [register frac32\\_t lval](#) ) [[inline](#)], [[static](#)]

The [MMAU\\_d\\_muls\\_dl](#) function multiplies 64-bit fractional value with 32-bit fractional value returning saturated 64-bit fractional product.

Parameters

<i>dval</i>	<a href="#">frac64_t</a> fractional value.
<i>lval</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac64\\_t](#) fractional value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation. If saturation occurs, the instruction sets the accumulation overflow (Q) and multiply or divide overflow (V) flags to 1 in the CSR. Otherwise, it clears the Q and V flags.

#### 20.5.63 static [frac64\\_t](#) MMAU\_d\_mula\_I ( register [frac32\\_t](#) *lval* ) [inline], [static]

The [MMAU\\_d\\_mula\\_I](#) function multiplies 32-bit fractional value with 64-bit fractional value stored in the A10 register of the MMAU returning a 64-bit fractional product.

Parameters

<i>lval</i>	<a href="#">frac32_t</a> fractional value.
-------------	--

Returns

[frac64\\_t](#) fractional value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.64 static [frac64\\_t](#) MMAU\_d\_mulas\_I ( register [frac32\\_t](#) *lval* ) [inline], [static]

The [MMAU\\_d\\_mulas\\_I](#) function multiplies 32-bit fractional value with 64-bit fractional value stored in the A10 register of the MMAU returning saturated 64-bit fractional product.

Parameters

<i>lval</i>	<a href="#">frac32_t</a> fractional value.
-------------	--

Returns

[frac64\\_t](#) fractional value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation. If saturation occurs, the instruction sets the accumulation overflow (Q) and multiply or divide overflow (V) flags to 1 in the CSR. Otherwise, it clears the Q and V flags.

#### 20.5.65 static [frac32\\_t](#) **MMAU\_I\_mul\_dl** ( [register frac64\\_t dval](#), [register frac32\\_t lval](#) ) [[inline](#)], [[static](#)]

The [MMAU\\_I\\_mul\\_dl](#) function multiplies 64-bit fractional value with 32-bit fractional value returning a 32-bit fractional product.

Parameters

<i>dval</i>	<a href="#">frac64_t</a> fractional value.
<i>lval</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac32\\_t](#) fractional value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.66 static [frac32\\_t](#) **MMAU\_I\_muls\_dl** ( [register frac64\\_t dval](#), [register frac32\\_t lval](#) ) [[inline](#)], [[static](#)]

The [MMAU\\_I\\_muls\\_dl](#) function multiplies 64-bit fractional value with 32-bit fractional value returning saturated 32-bit fractional product.

Parameters

<i>dval</i>	<a href="#">frac64_t</a> fractional value.
<i>lval</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac32\\_t](#) fractional value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation.

#### 20.5.67 **static frac32\_t MMAU\_I\_mula\_I( register frac32\_t *lval* ) [inline], [static]**

The [MMAU\\_I\\_mula\\_I](#) function multiplies 32-bit fractional value with 64-bit fractional value stored in the A10 register of the MMAU returning a 32-bit fractional product.

Parameters

<i>lval</i>	<a href="#">frac32_t</a> fractional value.
-------------	--

Returns

[frac32\\_t](#) fractional value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.68 **static frac32\_t MMAU\_I\_mulas\_I( register frac32\_t *lval* ) [inline], [static]**

The [MMAU\\_I\\_mulas\\_I](#) function multiplies 32-bit fractional value with 64-bit fractional value stored in the A10 register of the MMAU returning saturated 32-bit fractional product.

Parameters

<i>lval</i>	<a href="#">frac32_t</a> fractional value.
-------------	--

Returns

[frac32\\_t](#) fractional value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation.

#### 20.5.69 static [frac64\\_t](#) MMAU\_d\_mac\_ll ( [register frac32\\_t lval1](#), [register frac32\\_t lval2](#) ) [[inline](#)], [[static](#)]

The [MMAU\\_d\\_mac\\_ll](#) function multiplies two 32-bit fractional values and add product with value stored in the A10 register of the MMAU returning a 64-bit fractional A10 register value.

Parameters

<i>lval1</i>	<a href="#">frac32_t</a> fractional value.
<i>lval2</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac64\\_t](#) fractional value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.70 static [frac64\\_t](#) MMAU\_d\_macs\_ll ( [register frac32\\_t lval1](#), [register frac32\\_t lval2](#) ) [[inline](#)], [[static](#)]

The [MMAU\\_d\\_macs\\_ll](#) function multiplies two 32-bit fractional values and add product with value stored in the A10 register of the MMAU returning saturated 64-bit fractional A10 register value.

Parameters

<i>lval1</i>	<a href="#">frac32_t</a> fractional value.
<i>lval2</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac64\\_t](#) fractional value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation. If saturation occurs, the instruction sets the accumulation overflow (Q) and multiply or divide overflow (V) flags to 1 in the CSR. Otherwise, it clears the Q and V flags.

#### 20.5.71 static [frac64\\_t](#) MMAU\_d\_mac\_dl ( [register frac64\\_t dval](#), [register frac32\\_t lval](#) ) [[inline](#)], [[static](#)]

The [MMAU\\_d\\_mac\\_dl](#) function multiplies 64-bit fractional value with 32-bit fractional value and add product with value stored in the A10 register of the MMAU returning a 64-bit fractional A10 register value.

Parameters

<i>dval</i>	<a href="#">frac64_t</a> fractional value.
<i>lval</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac64\\_t](#) fractional value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.72 static [frac64\\_t](#) MMAU\_d\_macs\_dl ( [register frac64\\_t dval](#), [register frac32\\_t lval](#) ) [[inline](#)], [[static](#)]

The [MMAU\\_d\\_macs\\_dl](#) function multiplies 64-bit fractional value with 32-bit fractional value and add product with value stored in the A10 register of the MMAU returning saturated 64-bit fractional A10 register value.

Parameters

<i>dval</i>	<a href="#">frac64_t</a> fractional value.
<i>lval</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac64\\_t](#) fractional value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation. If saturation occurs, the instruction sets the accumulation overflow (Q) and multiply or divide overflow (V) flags to 1 in the CSR. Otherwise, it clears the Q and V flags.

#### 20.5.73 static [frac64\\_t](#) MMAU\_d\_maca\_dl ( [register frac64\\_t dval](#), [register frac32\\_t lval](#) ) [inline], [static]

The [MMAU\\_d\\_maca\\_dl](#) function multiplies 32-bit fractional value by value stored in the A10 register of the MMAU and add product with 64-bit fractional value returning a 64-bit fractional A10 register value.

Parameters

<i>dval</i>	<a href="#">frac64_t</a> fractional value.
<i>lval</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac64\\_t](#) fractional value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.74 static [frac64\\_t](#) MMAU\_d\_macas\_dl ( [register frac64\\_t dval](#), [register frac32\\_t lval](#) ) [inline], [static]

The [MMAU\\_d\\_macas\\_dl](#) function multiplies 32-bit fractional value by value stored in the A10 register of the MMAU and add product with 64-bit fractional value returning saturated 64-bit fractional A10 register value.

Parameters

<i>dval</i>	<a href="#">frac64_t</a> fractional value.
<i>lval</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac64\\_t](#) fractional value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation. If saturation occurs, the instruction sets the accumulation overflow (Q) and multiply or divide overflow (V) flags to 1 in the CSR. Otherwise, it clears the Q and V flags.

#### 20.5.75 static [frac32\\_t](#) **MMAU\_I\_mac\_ll** ( [register frac32\\_t lval1](#), [register frac32\\_t lval2](#) ) [[inline](#)], [[static](#)]

The [MMAU\\_I\\_mac\\_ll](#) function multiplies two 32-bit fractional values and add product with value stored in the A10 register of the MMAU returning a 32-bit fractional A1 register value.

Parameters

<i>lval1</i>	<a href="#">frac32_t</a> fractional value.
<i>lval2</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac32\\_t](#) fractional value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.76 static [frac32\\_t](#) **MMAU\_I\_macs\_ll** ( [register frac32\\_t lval1](#), [register frac32\\_t lval2](#) ) [[inline](#)], [[static](#)]

The [MMAU\\_I\\_macs\\_ll](#) function multiplies two 32-bit fractional values and add product with value stored in the A10 register of the MMAU returning saturated 32-bit fractional A1 register value.

Parameters

<i>lval1</i>	<a href="#">frac32_t</a> fractional value.
<i>lval2</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac32\\_t](#) fractional value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation.

#### 20.5.77 static [frac32\\_t](#) **MMAU\_I\_mac\_dl** ( [register frac64\\_t dval](#), [register frac32\\_t lval](#) ) [[inline](#)], [[static](#)]

The [MMAU\\_I\\_mac\\_dl](#) function multiplies 64-bit fractional value with 32-bit fractional value and add product with value stored in the A10 register of the MMAU returning a 32-bit fractional A1 register value.

Parameters

<i>dval</i>	<a href="#">frac64_t</a> fractional value.
<i>lval</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac32\\_t](#) fractional value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.78 static [frac32\\_t](#) **MMAU\_I\_macs\_dl** ( [register frac64\\_t dval](#), [register frac32\\_t lval](#) ) [[inline](#)], [[static](#)]

The [MMAU\\_I\\_macs\\_dl](#) function multiplies 64-bit fractional value with 32-bit fractional value and add product with value stored in the A10 register of the MMAU returning saturated 32-bit fractional A1 register value.

Parameters

<i>dval</i>	<a href="#">frac64_t</a> fractional value.
<i>lval</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac32\\_t](#) fractional value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation.

#### 20.5.79 **static frac32\_t MMAU\_I\_maca\_dl ( register frac64\_t *dval*, register frac32\_t *lval* ) [inline], [static]**

The [MMAU\\_1\\_maca\\_dl](#) function multiplies 32-bit fractional value by value stored in the A10 register of the MMAU and add product with 64-bit fractional value returning a 32-bit fractional A1 register value.

Parameters

<i>dval</i>	<a href="#">frac64_t</a> fractional value.
<i>lval</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac32\\_t](#) fractional value after multiply.

Note

Product is stored in A10 register of the MMAU for next computation.

#### 20.5.80 **static frac32\_t MMAU\_I\_macas\_dl ( register frac64\_t *dval*, register frac32\_t *lval* ) [inline], [static]**

The [MMAU\\_1\\_macas\\_dl](#) function multiplies 32-bit fractional value by value stored in the A10 register of the MMAU and add product with 64-bit fractional value returning saturated 32-bit fractional A1 register value.

Parameters

<i>dval</i>	<a href="#">frac64_t</a> fractional value.
<i>lval</i>	<a href="#">frac32_t</a> fractional value.

Returns

[frac32\\_t](#) fractional value after multiply.

Note

Saturated product is stored in A10 register of the MMAU for next computation.

#### 20.5.81 static [frac32\\_t](#) MMAU\_I\_div\_ll ( [register frac32\\_t](#) *Inum*, [register frac32\\_t](#) *Iden* ) [inline], [static]

The [MMAU\\_I\\_div\\_ll](#) function divides two 32-bit fractional values returning a 32-bit fractional quotient.

Parameters

<i>Inum</i>	<a href="#">frac32_t</a> fractional divisor value.
<i>Iden</i>	<a href="#">frac32_t</a> fractional dividend value.

Returns

[frac32\\_t](#) fractional quotient value.

Note

Quotient is stored in A1 register of the MMAU for next computation.

#### 20.5.82 static [frac32\\_t](#) MMAU\_I\_divs\_ll ( [register frac32\\_t](#) *Inum*, [register frac32\\_t](#) *Iden* ) [inline], [static]

The [MMAU\\_I\\_divs\\_ll](#) function divides two 32-bit fractional values returning a 32-bit fractional quotient.

Parameters

<i>lnum</i>	<a href="#">frac32_t</a> fractional divisor value.
<i>lden</i>	<a href="#">frac32_t</a> fractional dividend value.

Returns

[frac32\\_t](#) fractional quotient value.

Note

Saturated quotient is stored in A1 register of the MMAU for next computation. If saturation occurs, the instruction sets the accumulation overflow (Q) and multiply or divide overflow (V) flags to 1 in the CSR. Otherwise, it clears the Q and V flags.

#### 20.5.83 static [frac32\\_t](#) MMAU\_I\_divas\_I ( [register frac32\\_t lden](#) ) [inline], [[static](#)]

The [MMAU\\_I\\_divas\\_I](#) function divides 64-bit fractional value stored in the A10 register of the MMAU by 32-bit fractional value returning a saturated 32-bit fractional quotient.

Parameters

<i>lden</i>	<a href="#">frac32_t</a> fractional dividend value.
-------------	---

Returns

[frac32\\_t](#) fractional quotient value.

Note

Saturated quotient is stored in A1 register of the MMAU for next computation.

#### 20.5.84 static [frac64\\_t](#) MMAU\_d\_div\_dl ( [register frac64\\_t dnum](#), [register frac32\\_t lden](#) ) [inline], [[static](#)]

The [MMAU\\_d\\_div\\_dl](#) function divides 64-bit fractional value by 32-bit fractional value returning a 64-bit fractional quotient.

Parameters

<i>dnum</i>	<a href="#">frac64_t</a> fractional divisor value.
<i>lden</i>	<a href="#">frac32_t</a> fractional dividend value.

Returns

[frac64\\_t](#) fractional quotient value.

Note

Quotient is stored in A10 register of the MMAU for next computation.

#### 20.5.85 **static frac64\_t MMAU\_d\_divs\_dl ( register frac64\_t *dnum*, register frac32\_t *lden* ) [inline], [static]**

The [MMAU\\_d\\_divs\\_dl](#) function divides 64-bit fractional value by 32-bit fractional value returning a 64-bit fractional quotient.

Parameters

<i>dnum</i>	<a href="#">frac64_t</a> fractional divisor value.
<i>lden</i>	<a href="#">frac32_t</a> fractional dividend value.

Returns

[frac64\\_t](#) fractional quotient value.

Note

Saturated quotient is stored in A10 register of the MMAU for next computation. If saturation occurs, the instruction sets the accumulation overflow (Q) and multiply or divide overflow (V) flags to 1 in the CSR. Otherwise, it clears the Q and V flags.

#### 20.5.86 **static frac64\_t MMAU\_d\_diva\_l ( register frac32\_t *lden1* ) [inline], [static]**

The [MMAU\\_d\\_diva\\_l](#) function divides 32-bit fractional value stored in the A10 register of the MMAU by 32-bit fractional value returning a 64-bit fractional quotient.

Parameters

<i>lden1</i>	<a href="#">frac32_t</a> fractional dividend value.
--------------	---

Returns

[frac64\\_t](#) fractional quotient value.

Note

Quotient is stored in A10 register of the MMAU for next computation.

#### 20.5.87 **static frac64\_t MMAU\_d\_divas\_1( register frac32\_t *lden1* ) [inline], [static]**

The [MMAU\\_d\\_divas\\_1](#) function divides 32-bit fractional value stored in the A10 register of the MMAU by 32-bit fractional value returning a saturated 64-bit fractional quotient.

Parameters

<i>lden1</i>	<a href="#">frac32_t</a> fractional dividend value.
--------------	---

Returns

[frac64\\_t](#) fractional quotient value.

Note

Saturated quotient is stored in A10 register of the MMAU for next computation. If saturation occurs, the instruction sets the accumulation overflow (Q) and multiply or divide overflow (V) flags to 1 in the CSR. Otherwise, it clears the Q and V flags.

#### 20.5.88 **static frac32\_t MMAU\_l\_diva\_1( register frac32\_t *lden* ) [inline], [static]**

The [MMAU\\_l\\_diva\\_1](#) function divides 64-bit fractional value stored in the A10 register of the MMAU by 32-bit fractional value returning a 32-bit fractional quotient.

Parameters

<i>lden</i>	<a href="#">frac32_t</a> fractional dividend value.
-------------	---

Returns

[frac32\\_t](#) fractional quotient value.

Note

Quotient is stored in A1 register of the MMAU for next computation.

#### 20.5.89 **static frac32\_t MMAU\_I\_sqr\_I( register frac32\_t *lrad* ) [inline], [static]**

The [MMAU\\_I\\_sqr\\_I](#) function computes and returns a 32-bit fractional square root of the 32-bit fractional radicand.

Parameters

<i>lrad</i>	<a href="#">frac32_t</a> fractional radicand.
-------------	---

Returns

[frac32\\_t](#) fractional square root.

Note

Square root is stored in A1 register of the MMAU for next computation.

#### 20.5.90 **static frac32\_t MMAU\_I\_sqr\_d( register frac64\_t *drad* ) [inline], [static]**

The [MMAU\\_I\\_sqr\\_d](#) function computes and returns a 32-bit fractional square root of the 64-bit fractional radicand.

Parameters

<i>drad</i>	<a href="#">frac64_t</a> fractional radicand.
-------------	---

Returns

[frac32\\_t](#) fractional square root.

Note

Quotient is stored in A1 register of the MMAU for next computation.

### 20.5.91 static [frac32\\_t](#) **MMAU\_1\_sqra** ( void ) [inline], [static]

The [MMAU\\_1\\_sqra](#) function computes and returns a 32-bit fractional square root of the radicand stored in the A10 register of the MMAU.

Returns

[frac32\\_t](#) fractional square root.

Note

Quotient is stored in A1 register of the MMAU for next computation.

# Chapter 21

## PDB: Programmable Delay Block

### 21.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Programmable Delay Block (PDB) module of MCUXpresso SDK devices.

The PDB driver includes a basic PDB counter, trigger generators for ADC, DAC, and pulse-out.

The basic PDB counter can be used as a general programmable timer with an interrupt. The counter increases automatically with the divided clock signal after it is triggered to start by an external trigger input or the software trigger. There are "milestones" for the output trigger event. When the counter is equal to any of these "milestones", the corresponding trigger is generated and sent out to other modules. These "milestones" are for the following events.

- Counter delay interrupt, which is the interrupt for the PDB module
- ADC pre-trigger to trigger the ADC conversion
- DAC interval trigger to trigger the DAC buffer and move the buffer read pointer
- Pulse-out triggers to generate a single or rising and falling edges, which can be assembled to a window.

The "milestone" values have a flexible load mode. To call the APIs to set these value is equivalent to writing data to their buffer. The loading event occurs as the load mode describes. This design ensures that all "milestones" can be updated at the same time.

### 21.2 Typical use case

#### 21.2.1 Working as basic PDB counter with a PDB interrupt.

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

#### 21.2.2 Working with an additional trigger. The ADC trigger is used as an example.

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

### Data Structures

- struct [pdb\\_config\\_t](#)  
*PDB module configuration. [More...](#)*
- struct [pdb\\_adc\\_pretrigger\\_config\\_t](#)  
*PDB ADC Pre-trigger configuration. [More...](#)*
- struct [pdb\\_dac\\_trigger\\_config\\_t](#)  
*PDB DAC trigger configuration. [More...](#)*

## Enumerations

- enum `_pdb_status_flags` {
   
  `kPDB_LoadOKFlag` = `PDB_SC_LDOK_MASK`,
   
  `kPDB_DelayEventFlag` = `PDB_SC_PDBIF_MASK` }
   
*PDB flags.*
- enum `_pdb_adc_pretrigger_flags` {
   
  `kPDB_ADCPreTriggerChannel0Flag` = `PDB_S_CF(1U << 0)`,
   
  `kPDB_ADCPreTriggerChannel1Flag` = `PDB_S_CF(1U << 1)`,
   
  `kPDB_ADCPreTriggerChannel2Flag` = `PDB_S_CF(1U << 2)`,
   
  `kPDB_ADCPreTriggerChannel3Flag` = `PDB_S_CF(1U << 3)`,
   
  `kPDB_ADCPreTriggerChannel0ErrorFlag` = `PDB_S_ERR(1U << 0)`,
   
  `kPDB_ADCPreTriggerChannel1ErrorFlag` = `PDB_S_ERR(1U << 1)`,
   
  `kPDB_ADCPreTriggerChannel2ErrorFlag` = `PDB_S_ERR(1U << 2)`,
   
  `kPDB_ADCPreTriggerChannel3ErrorFlag` = `PDB_S_ERR(1U << 3)` }
   
*PDB ADC PreTrigger channel flags.*
- enum `_pdb_interrupt_enable` {
   
  `kPDB_SequenceErrorInterruptEnable` = `PDB_SC_PDDEIE_MASK`,
   
  `kPDB_DelayInterruptEnable` = `PDB_SC_PDBIE_MASK` }
   
*PDB buffer interrupts.*
- enum `pdb_load_value_mode_t` {
   
  `kPDB_LoadValueImmediately` = `0U`,
   
  `kPDB_LoadValueOnCounterOverflow` = `1U`,
   
  `kPDB_LoadValueOnTriggerInput` = `2U`,
   
  `kPDB_LoadValueOnCounterOverflowOrTriggerInput` = `3U` }
   
*PDB load value mode.*
- enum `pdb_prescaler_divider_t` {
   
  `kPDB_PrescalerDivider1` = `0U`,
   
  `kPDB_PrescalerDivider2` = `1U`,
   
  `kPDB_PrescalerDivider4` = `2U`,
   
  `kPDB_PrescalerDivider8` = `3U`,
   
  `kPDB_PrescalerDivider16` = `4U`,
   
  `kPDB_PrescalerDivider32` = `5U`,
   
  `kPDB_PrescalerDivider64` = `6U`,
   
  `kPDB_PrescalerDivider128` = `7U` }
   
*Prescaler divider.*
- enum `pdb_divider_multiplication_factor_t` {
   
  `kPDB_DividerMultiplicationFactor1` = `0U`,
   
  `kPDB_DividerMultiplicationFactor10` = `1U`,
   
  `kPDB_DividerMultiplicationFactor20` = `2U`,
   
  `kPDB_DividerMultiplicationFactor40` = `3U` }
   
*Multiplication factor select for prescaler.*
- enum `pdb_trigger_input_source_t` {

```

kPDB_TriggerInput0 = 0U,
kPDB_TriggerInput1 = 1U,
kPDB_TriggerInput2 = 2U,
kPDB_TriggerInput3 = 3U,
kPDB_TriggerInput4 = 4U,
kPDB_TriggerInput5 = 5U,
kPDB_TriggerInput6 = 6U,
kPDB_TriggerInput7 = 7U,
kPDB_TriggerInput8 = 8U,
kPDB_TriggerInput9 = 9U,
kPDB_TriggerInput10 = 10U,
kPDB_TriggerInput11 = 11U,
kPDB_TriggerInput12 = 12U,
kPDB_TriggerInput13 = 13U,
kPDB_TriggerInput14 = 14U,
kPDB_TriggerSoftware = 15U }

```

*Trigger input source.*

- enum `pdb_adc_trigger_channel_t` {
 

```

kPDB_ADCTriggerChannel0 = 0U,
kPDB_ADCTriggerChannel1 = 1U,
kPDB_ADCTriggerChannel2 = 2U,
kPDB_ADCTriggerChannel3 = 3U }
```

*List of PDB ADC trigger channels.*

- enum `pdb_adc_pretrigger_t` {
 

```

kPDB_ADCPreTrigger0 = 0U,
kPDB_ADCPreTrigger1 = 1U,
kPDB_ADCPreTrigger2 = 2U,
kPDB_ADCPreTrigger3 = 3U,
kPDB_ADCPreTrigger4 = 4U,
kPDB_ADCPreTrigger5 = 5U,
kPDB_ADCPreTrigger6 = 6U,
kPDB_ADCPreTrigger7 = 7U }
```

*List of PDB ADC pretrigger.*

- enum `pdb_dac_trigger_channel_t` {
 

```

kPDB_DACTriggerChannel0 = 0U,
kPDB_DACTriggerChannel1 = 1U }
```

*List of PDB DAC trigger channels.*

- enum `pdb_pulse_out_trigger_channel_t` {
 

```

kPDB_PulseOutTriggerChannel0 = 0U,
kPDB_PulseOutTriggerChannel1 = 1U,
kPDB_PulseOutTriggerChannel2 = 2U,
kPDB_PulseOutTriggerChannel3 = 3U }
```

*List of PDB pulse out trigger channels.*

- enum `pdb_pulse_out_channel_mask_t` {

```
kPDB_PulseOutChannel0Mask = (1U << 0U),
kPDB_PulseOutChannel1Mask = (1U << 1U),
kPDB_PulseOutChannel2Mask = (1U << 2U),
kPDB_PulseOutChannel3Mask = (1U << 3U) }
```

*List of PDB pulse out trigger channels mask.*

## Driver version

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

## Initialization

- void **PDB\_Init** (PDB\_Type \*base, const **pdb\_config\_t** \*config)  
*Initializes the PDB module.*
- void **PDB\_Deinit** (PDB\_Type \*base)  
*De-initializes the PDB module.*
- void **PDB\_GetDefaultConfig** (**pdb\_config\_t** \*config)  
*Initializes the PDB user configuration structure.*
- static void **PDB\_Enable** (PDB\_Type \*base, bool enable)  
*Enables the PDB module.*

## Basic Counter

- static void **PDB\_DoSoftwareTrigger** (PDB\_Type \*base)  
*Triggers the PDB counter by software.*
- static void **PDB\_DoLoadValues** (PDB\_Type \*base)  
*Loads the counter values.*
- static void **PDB\_EnableDMA** (PDB\_Type \*base, bool enable)  
*Enables the DMA for the PDB module.*
- static void **PDB\_EnableInterrupts** (PDB\_Type \*base, uint32\_t mask)  
*Enables the interrupts for the PDB module.*
- static void **PDB\_DisableInterrupts** (PDB\_Type \*base, uint32\_t mask)  
*Disables the interrupts for the PDB module.*
- static uint32\_t **PDB\_GetStatusFlags** (PDB\_Type \*base)  
*Gets the status flags of the PDB module.*
- static void **PDB\_ClearStatusFlags** (PDB\_Type \*base, uint32\_t mask)  
*Clears the status flags of the PDB module.*
- static void **PDB\_SetModulusValue** (PDB\_Type \*base, uint32\_t value)  
*Specifies the counter period.*
- static uint32\_t **PDB\_GetCounterValue** (PDB\_Type \*base)  
*Gets the PDB counter's current value.*
- static void **PDB\_SetCounterDelayValue** (PDB\_Type \*base, uint32\_t value)  
*Sets the value for the PDB counter delay event.*

## ADC Pre-trigger

- static void **PDB\_SetADCPreTriggerConfig** (PDB\_Type \*base, **pdb\_adc\_trigger\_channel\_t** channel, **pdb\_adc\_pretrigger\_config\_t** \*config)  
*Configures the ADC pre-trigger in the PDB module.*

- static void **PDB\_SetADCPreTriggerDelayValue** (PDB\_Type \*base, **pdb\_adc\_trigger\_channel\_t** channel, **pdb\_adc\_pretrigger\_t** pretriggerNumber, uint32\_t value)  
*Sets the value for the ADC pre-trigger delay event.*
- static uint32\_t **PDB\_GetADCPreTriggerStatusFlags** (PDB\_Type \*base, **pdb\_adc\_trigger\_channel\_t** channel)  
*Gets the ADC pre-trigger's status flags.*
- static void **PDB\_ClearADCPreTriggerStatusFlags** (PDB\_Type \*base, **pdb\_adc\_trigger\_channel\_t** channel, uint32\_t mask)  
*Clears the ADC pre-trigger status flags.*

## Pulse-Out Trigger

- static void **PDB\_EnablePulseOutTrigger** (PDB\_Type \*base, **pdb\_pulse\_out\_channel\_mask\_t** channelMask, bool enable)  
*Enables the pulse out trigger channels.*
- static void **PDB\_SetPulseOutTriggerDelayValue** (PDB\_Type \*base, **pdb\_pulse\_out\_trigger\_channel\_t** channel, uint32\_t value1, uint32\_t value2)  
*Sets event values for the pulse out trigger.*

## 21.3 Data Structure Documentation

### 21.3.1 struct **pdb\_config\_t**

#### Data Fields

- **pdb\_load\_value\_mode\_t** **loadValueMode**  
*Select the load value mode.*
- **pdb\_prescaler\_divider\_t** **prescalerDivide**  
*Select the prescaler divider.*
- **pdb\_divider\_multiplication\_factor\_t** **dividerMultiplicationFactor**  
*Multiplication factor select for prescaler.*
- **pdb\_trigger\_input\_source\_t** **triggerInputSource**  
*Select the trigger input source.*
- bool **enableContinuousMode**  
*Enable the PDB operation in Continuous mode.*

#### Field Documentation

- (1) **pdb\_load\_value\_mode\_t** **pdb\_config\_t::loadValueMode**
- (2) **pdb\_prescaler\_divider\_t** **pdb\_config\_t::prescalerDivide**
- (3) **pdb\_divider\_multiplication\_factor\_t** **pdb\_config\_t::dividerMultiplicationFactor**
- (4) **pdb\_trigger\_input\_source\_t** **pdb\_config\_t::triggerInputSource**
- (5) bool **pdb\_config\_t::enableContinuousMode**

### 21.3.2 struct pdb\_adc\_pretrigger\_config\_t

#### Data Fields

- `uint32_t enablePreTriggerMask`  
*PDB Channel Pre-trigger Enable.*
- `uint32_t enableOutputMask`  
*PDB Channel Pre-trigger Output Select.*
- `uint32_t enableBackToBackOperationMask`  
*PDB Channel pre-trigger Back-to-Back Operation Enable.*

#### Field Documentation

(1) `uint32_t pdb_adc_pretrigger_config_t::enablePreTriggerMask`

(2) `uint32_t pdb_adc_pretrigger_config_t::enableOutputMask`

PDB channel's corresponding pre-trigger asserts when the counter reaches the channel delay register.

(3) `uint32_t pdb_adc_pretrigger_config_t::enableBackToBackOperationMask`

Back-to-back operation enables the ADC conversions complete to trigger the next PDB channel pre-trigger and trigger output, so that the ADC conversions can be triggered on next set of configuration and results registers.

### 21.3.3 struct pdb\_dac\_trigger\_config\_t

#### Data Fields

- `bool enableExternalTriggerInput`  
*Enables the external trigger for DAC interval counter.*
- `bool enableIntervalTrigger`  
*Enables the DAC interval trigger.*

#### Field Documentation

(1) `bool pdb_dac_trigger_config_t::enableExternalTriggerInput`

(2) `bool pdb_dac_trigger_config_t::enableIntervalTrigger`

## 21.4 Macro Definition Documentation

21.4.1 `#define FSL_PDB_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))`

## 21.5 Enumeration Type Documentation

### 21.5.1 enum \_pdb\_status\_flags

Enumerator

***kPDB\_LoadOKFlag*** This flag is automatically cleared when the values in buffers are loaded into the internal registers after the LDOK bit is set or the PDBEN is cleared.

***kPDB\_DelayEventFlag*** PDB timer delay event flag.

### 21.5.2 enum \_pdb\_adc\_pretrigger\_flags

Enumerator

***kPDB\_ADCPreTriggerChannel0Flag*** Pre-trigger 0 flag.

***kPDB\_ADCPreTriggerChannel1Flag*** Pre-trigger 1 flag.

***kPDB\_ADCPreTriggerChannel2Flag*** Pre-trigger 2 flag.

***kPDB\_ADCPreTriggerChannel3Flag*** Pre-trigger 3 flag.

***kPDB\_ADCPreTriggerChannel0ErrorFlag*** Pre-trigger 0 Error.

***kPDB\_ADCPreTriggerChannel1ErrorFlag*** Pre-trigger 1 Error.

***kPDB\_ADCPreTriggerChannel2ErrorFlag*** Pre-trigger 2 Error.

***kPDB\_ADCPreTriggerChannel3ErrorFlag*** Pre-trigger 3 Error.

### 21.5.3 enum \_pdb\_interrupt\_enable

Enumerator

***kPDB\_SequenceErrorInterruptEnable*** PDB sequence error interrupt enable.

***kPDB\_DelayInterruptEnable*** PDB delay interrupt enable.

### 21.5.4 enum pdb\_load\_value\_mode\_t

Selects the mode to load the internal values after doing the load operation (write 1 to PDBx\_SC[LDOK]). These values are for the following operations.

- PDB counter (PDBx\_MOD, PDBx\_IDLY)
- ADC trigger (PDBx\_CHnDLYm)
- DAC trigger (PDBx\_DACINTx)
- CMP trigger (PDBx\_POyDLY)

Enumerator

***kPDB\_LoadValueImmediately*** Load immediately after 1 is written to LDOK.

***kPDB\_LoadValueOnCounterOverflow*** Load when the PDB counter overflows (reaches the MOD register value).

***kPDB\_LoadValueOnTriggerInput*** Load a trigger input event is detected.

***kPDB\_LoadValueOnCounterOverflowOrTriggerInput*** Load either when the PDB counter overflows or a trigger input is detected.

## 21.5.5 enum pdb\_prescaler\_divider\_t

Counting uses the peripheral clock divided by multiplication factor selected by times of MULT.

Enumerator

***kPDB\_PrescalerDivider1*** Divider x1.

***kPDB\_PrescalerDivider2*** Divider x2.

***kPDB\_PrescalerDivider4*** Divider x4.

***kPDB\_PrescalerDivider8*** Divider x8.

***kPDB\_PrescalerDivider16*** Divider x16.

***kPDB\_PrescalerDivider32*** Divider x32.

***kPDB\_PrescalerDivider64*** Divider x64.

***kPDB\_PrescalerDivider128*** Divider x128.

## 21.5.6 enum pdb\_divider\_multiplication\_factor\_t

Selects the multiplication factor of the prescaler divider for the counter clock.

Enumerator

***kPDB\_DividerMultiplicationFactor1*** Multiplication factor is 1.

***kPDB\_DividerMultiplicationFactor10*** Multiplication factor is 10.

***kPDB\_DividerMultiplicationFactor20*** Multiplication factor is 20.

***kPDB\_DividerMultiplicationFactor40*** Multiplication factor is 40.

## 21.5.7 enum pdb\_trigger\_input\_source\_t

Selects the trigger input source for the PDB. The trigger input source can be internal or external (EXTRG pin), or the software trigger. See chip configuration details for the actual PDB input trigger connections.

Enumerator

***kPDB\_TriggerInput0*** Trigger-In 0.

***kPDB\_TriggerInput1*** Trigger-In 1.

<i>kPDB_TriggerInput2</i>	Trigger-In 2.
<i>kPDB_TriggerInput3</i>	Trigger-In 3.
<i>kPDB_TriggerInput4</i>	Trigger-In 4.
<i>kPDB_TriggerInput5</i>	Trigger-In 5.
<i>kPDB_TriggerInput6</i>	Trigger-In 6.
<i>kPDB_TriggerInput7</i>	Trigger-In 7.
<i>kPDB_TriggerInput8</i>	Trigger-In 8.
<i>kPDB_TriggerInput9</i>	Trigger-In 9.
<i>kPDB_TriggerInput10</i>	Trigger-In 10.
<i>kPDB_TriggerInput11</i>	Trigger-In 11.
<i>kPDB_TriggerInput12</i>	Trigger-In 12.
<i>kPDB_TriggerInput13</i>	Trigger-In 13.
<i>kPDB_TriggerInput14</i>	Trigger-In 14.
<i>kPDB_TriggerSoftware</i>	Trigger-In 15, software trigger.

### 21.5.8 enum pdb\_adc\_trigger\_channel\_t

Note

Actual number of available channels is SoC dependent

Enumerator

<i>kPDB_ADCTriggerChannel0</i>	PDB ADC trigger channel number 0.
<i>kPDB_ADCTriggerChannel1</i>	PDB ADC trigger channel number 1.
<i>kPDB_ADCTriggerChannel2</i>	PDB ADC trigger channel number 2.
<i>kPDB_ADCTriggerChannel3</i>	PDB ADC trigger channel number 3.

### 21.5.9 enum pdb\_adc\_pretrigger\_t

Note

Actual number of available pretrigger channels is SoC dependent

Enumerator

<i>kPDB_ADCPreTrigger0</i>	PDB ADC pretrigger number 0.
<i>kPDB_ADCPreTrigger1</i>	PDB ADC pretrigger number 1.
<i>kPDB_ADCPreTrigger2</i>	PDB ADC pretrigger number 2.
<i>kPDB_ADCPreTrigger3</i>	PDB ADC pretrigger number 3.
<i>kPDB_ADCPreTrigger4</i>	PDB ADC pretrigger number 4.
<i>kPDB_ADCPreTrigger5</i>	PDB ADC pretrigger number 5.
<i>kPDB_ADCPreTrigger6</i>	PDB ADC pretrigger number 6.
<i>kPDB_ADCPreTrigger7</i>	PDB ADC pretrigger number 7.

**21.5.10 enum pdb\_dac\_trigger\_channel\_t**

Note

Actual number of available channels is SoC dependent

Enumerator

***kPDB\_DACTriggerChannel0*** PDB DAC trigger channel number 0.  
***kPDB\_DACTriggerChannel1*** PDB DAC trigger channel number 1.

**21.5.11 enum pdb\_pulse\_out\_trigger\_channel\_t**

Note

Actual number of available channels is SoC dependent

Enumerator

***kPDB\_PulseOutTriggerChannel0*** PDB pulse out trigger channel number 0.  
***kPDB\_PulseOutTriggerChannel1*** PDB pulse out trigger channel number 1.  
***kPDB\_PulseOutTriggerChannel2*** PDB pulse out trigger channel number 2.  
***kPDB\_PulseOutTriggerChannel3*** PDB pulse out trigger channel number 3.

**21.5.12 enum pdb\_pulse\_out\_channel\_mask\_t**

Note

Actual number of available channels mask is SoC dependent

Enumerator

***kPDB\_PulseOutChannel0Mask*** PDB pulse out trigger channel number 0 mask.  
***kPDB\_PulseOutChannel1Mask*** PDB pulse out trigger channel number 1 mask.  
***kPDB\_PulseOutChannel2Mask*** PDB pulse out trigger channel number 2 mask.  
***kPDB\_PulseOutChannel3Mask*** PDB pulse out trigger channel number 3 mask.

**21.6 Function Documentation****21.6.1 void PDB\_Init ( PDB\_Type \* *base*, const pdb\_config\_t \* *config* )**

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

- Enable the clock for PDB instance.
- Configure the PDB module.
- Enable the PDB module.

## Parameters

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

**21.6.2 void PDB\_Deinit ( PDB\_Type \* *base* )**

## Parameters

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

**21.6.3 void PDB\_GetDefaultConfig ( pdb\_config\_t \* *config* )**

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

```
* config->loadValueMode = kPDB_LoadValueImmediately;
* config->prescalerDivider = kPDB_PrescalerDivider1;
* config->dividerMultiplicationFactor = kPDB_DividerMultiplicationFactor1
  ;
* config->triggerInputSource = kPDB_TriggerSoftware;
* config->enableContinuousMode = false;
*
```

## Parameters

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

**21.6.4 static void PDB\_Enable ( PDB\_Type \* *base*, bool *enable* ) [inline], [static]**

## Parameters

<i>base</i>	PDB peripheral base address.
<i>enable</i>	Enable the module or not.

**21.6.5 static void PDB\_DoSoftwareTrigger ( PDB\_Type \* *base* ) [inline], [static]**

Parameters

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

#### 21.6.6 static void PDB\_DoLoadValues ( PDB\_Type \* *base* ) [inline], [static]

This function loads the counter values from the internal buffer. See "pdb\_load\_value\_mode\_t" about PDB's load mode.

Parameters

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

#### 21.6.7 static void PDB\_EnableDMA ( PDB\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	PDB peripheral base address.
<i>enable</i>	Enable the feature or not.

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

Parameters

<i>base</i>	PDB peripheral base address.
<i>mask</i>	Mask value for interrupts. See "_pdb_interrupt_enable".

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

Parameters

<i>base</i>	PDB peripheral base address.
<i>mask</i>	Mask value for interrupts. See "_pdb_interrupt_enable".

#### 21.6.10 static uint32\_t PDB\_GetStatusFlags ( PDB\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

Mask value for asserted flags. See "\_pdb\_status\_flags".

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

Parameters

<i>base</i>	PDB peripheral base address.
<i>mask</i>	Mask value of flags. See "_pdb_status_flags".

#### 21.6.12 static void PDB\_SetModulusValue ( PDB\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Parameters

<i>base</i>	PDB peripheral base address.
<i>value</i>	Setting value for the modulus. 16-bit is available.

#### 21.6.13 static uint32\_t PDB\_GetCounterValue ( PDB\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

PDB counter's current value.

#### 21.6.14 static void PDB\_SetCounterDelayValue ( PDB\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Parameters

<i>base</i>	PDB peripheral base address.
<i>value</i>	Setting value for PDB counter delay event. 16-bit is available.

#### 21.6.15 static void PDB\_SetADCPreTriggerConfig ( PDB\_Type \* *base*, pdb\_adc\_trigger\_channel\_t *channel*, pdb\_adc\_pretrigger\_config\_t \* *config* ) [inline], [static]

Parameters

<i>base</i>	PDB peripheral base address.
<i>channel</i>	Channel index for ADC instance.
<i>config</i>	Pointer to the configuration structure. See "pdb_adc_pretrigger_config_t".

#### 21.6.16 static void PDB\_SetADCPreTriggerDelayValue ( PDB\_Type \* *base*, pdb\_adc\_trigger\_channel\_t *channel*, pdb\_adc\_pretrigger\_t *pretriggerNumber*, uint32\_t *value* ) [inline], [static]

This function sets the value for ADC pre-trigger delay event. It specifies the delay value for the channel's corresponding pre-trigger. The pre-trigger asserts when the PDB counter is equal to the set value.

Parameters

<i>base</i>	PDB peripheral base address.
<i>channel</i>	Channel index for ADC instance.
<i>pretrigger-Number</i>	Channel group index for ADC instance.
<i>value</i>	Setting value for ADC pre-trigger delay event. 16-bit is available.

**21.6.17 static uint32\_t PDB\_GetADCPreTriggerStatusFlags ( PDB\_Type \* *base*, pdb\_adc\_trigger\_channel\_t *channel* ) [inline], [static]**

Parameters

<i>base</i>	PDB peripheral base address.
<i>channel</i>	Channel index for ADC instance.

Returns

Mask value for asserted flags. See "\_pdb\_adc\_pretrigger\_flags".

**21.6.18 static void PDB\_ClearADCPreTriggerStatusFlags ( PDB\_Type \* *base*, pdb\_adc\_trigger\_channel\_t *channel*, uint32\_t *mask* ) [inline], [static]**

Parameters

<i>base</i>	PDB peripheral base address.
<i>channel</i>	Channel index for ADC instance.
<i>mask</i>	Mask value for flags. See "_pdb_adc_pretrigger_flags".

**21.6.19 static void PDB\_EnablePulseOutTrigger ( PDB\_Type \* *base*, pdb\_pulse\_out\_channel\_mask\_t *channelMask*, bool *enable* ) [inline], [static]**

Parameters

<i>base</i>	PDB peripheral base address.
<i>channelMask</i>	Channel mask value for multiple pulse out trigger channel.
<i>enable</i>	Whether the feature is enabled or not.

**21.6.20 static void PDB\_SetPulseOutTriggerDelayValue ( PDB\_Type \* *base*,  
                  pdb\_pulse\_out\_trigger\_channel\_t *channel*, uint32\_t *value1*, uint32\_t  
                  *value2* ) [inline], [static]**

This function is used to set event values for the pulse output trigger. These pulse output trigger delay values specify the delay for the PDB Pulse-out. Pulse-out goes high when the PDB counter is equal to the pulse output high value (*value1*). Pulse-out goes low when the PDB counter is equal to the pulse output low value (*value2*).

Parameters

<i>base</i>	PDB peripheral base address.
<i>channel</i>	Channel index for pulse out trigger channel.
<i>value1</i>	Setting value for pulse out high.
<i>value2</i>	Setting value for pulse out low.

# Chapter 22

## PIT: Periodic Interrupt Timer

### 22.1 Overview

The MCUXpresso SDK provides a driver for the Periodic Interrupt Timer (PIT) of MCUXpresso SDK devices.

### 22.2 Function groups

The PIT driver supports operating the module as a time counter.

#### 22.2.1 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.

#### 22.2.2 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.

#### 22.2.3 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.

## 22.2.4 Status

Provides functions to get and clear the PIT status.

## 22.2.5 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

## 22.3 Typical use case

### 22.3.1 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.*

## 22.4 Data Structure Documentation

### 22.4.1 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*

## 22.5 Enumeration Type Documentation

### 22.5.1 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.

### 22.5.2 enum pit\_interrupt\_enable\_t

Enumerator

- kPIT\_TimerInterruptEnable*** Timer interrupt enable.

### 22.5.3 enum pit\_status\_flags\_t

Enumerator

- kPIT\_TimerFlag*** Timer flag.

## 22.6 Function Documentation

### 22.6.1 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

## 22.6.2 void PIT\_Deinit ( PIT\_Type \* *base* )

Parameters

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

## 22.6.3 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.
---------------	---

## 22.6.4 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.

## 22.6.5 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>

## 22.6.6 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>

## 22.6.7 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](#)

22.6.8 **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](#)

### 22.6.9 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>

### 22.6.10 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
--------------	--------------------------------

### 22.6.11 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

### 22.6.12 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.

### 22.6.13 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.

### 22.6.14 `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 23

## PMC: Power Management Controller

### 23.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Power Management Controller (PMC) module of MCUXpresso SDK devices. The PMC module contains internal voltage regulator, power on reset, low-voltage detect system, and high-voltage detect system.

### Data Structures

- struct [pmc\\_low\\_volt\\_detect\\_config\\_t](#)  
*Low-voltage Detect Configuration Structure. [More...](#)*
- struct [pmc\\_low\\_volt\\_warning\\_config\\_t](#)  
*Low-voltage Warning Configuration Structure. [More...](#)*
- struct [pmc\\_bandgap\\_buffer\\_config\\_t](#)  
*Bandgap Buffer configuration. [More...](#)*

### Enumerations

- enum [pmc\\_low\\_volt\\_detect\\_volt\\_select\\_t](#) {  
  kPMC\_LowVoltDetectLowTrip = 0U,  
  kPMC\_LowVoltDetectHighTrip = 1U }  
*Low-voltage Detect Voltage Select.*
- enum [pmc\\_low\\_volt\\_warning\\_volt\\_select\\_t](#) {  
  kPMC\_LowVoltWarningLowTrip = 0U,  
  kPMC\_LowVoltWarningMid1Trip = 1U,  
  kPMC\_LowVoltWarningMid2Trip = 2U,  
  kPMC\_LowVoltWarningHighTrip = 3U }  
*Low-voltage Warning Voltage Select.*

### Driver version

- #define [FSL\\_PMC\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 3))  
*PMC driver version.*

### Power Management Controller Control APIs

- void [PMC\\_ConfigureLowVoltDetect](#) (PMC\_Type \*base, const [pmc\\_low\\_volt\\_detect\\_config\\_t](#) \*config)  
*Configures the low-voltage detect setting.*
- static bool [PMC\\_GetLowVoltDetectFlag](#) (PMC\_Type \*base)  
*Gets the Low-voltage Detect Flag status.*
- static void [PMC\\_ClearLowVoltDetectFlag](#) (PMC\_Type \*base)  
*Acknowledges clearing the Low-voltage Detect flag.*

- void **PMC\_ConfigureLowVoltWarning** (PMC\_Type \*base, const **pmc\_low\_volt\_warning\_config\_t** \*config)  
*Configures the low-voltage warning setting.*
- static bool **PMC\_GetLowVoltWarningFlag** (PMC\_Type \*base)  
*Gets the Low-voltage Warning Flag status.*
- static void **PMC\_ClearLowVoltWarningFlag** (PMC\_Type \*base)  
*Acknowledges the Low-voltage Warning flag.*
- void **PMC\_ConfigureBandgapBuffer** (PMC\_Type \*base, const **pmc\_bandgap\_buffer\_config\_t** \*config)  
*Configures the PMC bandgap.*
- static bool **PMC\_GetPeriphIOIsolationFlag** (PMC\_Type \*base)  
*Gets the acknowledge Peripherals and I/O pads isolation flag.*
- static void **PMC\_ClearPeriphIOIsolationFlag** (PMC\_Type \*base)  
*Acknowledges the isolation flag to Peripherals and I/O pads.*
- static bool **PMC\_IsRegulatorInRunRegulation** (PMC\_Type \*base)  
*Gets the regulator regulation status.*

## 23.2 Data Structure Documentation

### 23.2.1 struct pmc\_low\_volt\_detect\_config\_t

#### Data Fields

- bool **enableInt**  
*Enable interrupt when Low-voltage detect.*
- bool **enableReset**  
*Enable system reset when Low-voltage detect.*
- **pmc\_low\_volt\_detect\_volt\_select\_t** **voltSelect**  
*Low-voltage detect trip point voltage selection.*

### 23.2.2 struct pmc\_low\_volt\_warning\_config\_t

#### Data Fields

- bool **enableInt**  
*Enable interrupt when low-voltage warning.*
- **pmc\_low\_volt\_warning\_volt\_select\_t** **voltSelect**  
*Low-voltage warning trip point voltage selection.*

### 23.2.3 struct pmc\_bandgap\_buffer\_config\_t

#### Data Fields

- bool **enable**  
*Enable bandgap buffer.*
- bool **enableInLowPowerMode**

*Enable bandgap buffer in low-power mode.*

### Field Documentation

- (1) `bool pmc_bandgap_buffer_config_t::enable`
- (2) `bool pmc_bandgap_buffer_config_t::enableInLowPowerMode`

## 23.3 Macro Definition Documentation

### 23.3.1 `#define FSL_PMC_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))`

Version 2.0.3.

## 23.4 Enumeration Type Documentation

### 23.4.1 `enum pmc_low_volt_detect_volt_select_t`

Enumerator

- kPMC\_LowVoltDetectLowTrip* Low-trip point selected (VLVD = VLVDL )
- kPMC\_LowVoltDetectHighTrip* High-trip point selected (VLVD = VLVDH )

### 23.4.2 `enum pmc_low_volt_warning_volt_select_t`

Enumerator

- kPMC\_LowVoltWarningLowTrip* Low-trip point selected (VLVW = VLVW1)
- kPMC\_LowVoltWarningMid1Trip* Mid 1 trip point selected (VLVW = VLVW2)
- kPMC\_LowVoltWarningMid2Trip* Mid 2 trip point selected (VLVW = VLVW3)
- kPMC\_LowVoltWarningHighTrip* High-trip point selected (VLVW = VLVW4)

## 23.5 Function Documentation

### 23.5.1 `void PMC_ConfigureLowVoltDetect ( PMC_Type * base, const pmc_low_volt_detect_config_t * config )`

This function configures the low-voltage detect setting, including the trip point voltage setting, enables or disables the interrupt, enables or disables the system reset.

Parameters

<i>base</i>	PMC peripheral base address.
<i>config</i>	Low-voltage detect configuration structure.

### 23.5.2 static bool PMC\_GetLowVoltDetectFlag ( **PMC\_Type** \* *base* ) [inline], [static]

This function reads the current LVDF status. If it returns 1, a low-voltage event is detected.

Parameters

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

Returns

Current low-voltage detect flag

- true: Low-voltage detected
- false: Low-voltage not detected

### 23.5.3 static void PMC\_ClearLowVoltDetectFlag ( **PMC\_Type** \* *base* ) [inline], [static]

This function acknowledges the low-voltage detection errors (write 1 to clear LVDF).

Parameters

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

### 23.5.4 void PMC\_ConfigureLowVoltWarning ( **PMC\_Type** \* *base*, const **pmc\_low\_volt\_warning\_config\_t** \* *config* )

This function configures the low-voltage warning setting, including the trip point voltage setting and enabling or disabling the interrupt.

Parameters

<i>base</i>	PMC peripheral base address.
<i>config</i>	Low-voltage warning configuration structure.

### 23.5.5 static bool PMC\_GetLowVoltWarningFlag ( **PMC\_Type** \* *base* ) [inline], [static]

This function polls the current LVWF status. When 1 is returned, it indicates a low-voltage warning event. LVWF is set when V Supply transitions below the trip point or after reset and V Supply is already below the V LVW.

Parameters

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

Returns

Current LVWF status

- true: Low-voltage Warning Flag is set.
- false: the Low-voltage Warning does not happen.

### 23.5.6 static void PMC\_ClearLowVoltWarningFlag ( **PMC\_Type** \* *base* ) [inline], [static]

This function acknowledges the low voltage warning errors (write 1 to clear LVWF).

Parameters

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

### 23.5.7 void PMC\_ConfigureBandgapBuffer ( **PMC\_Type** \* *base*, const **pmc\_bandgap\_buffer\_config\_t** \* *config* )

This function configures the PMC bandgap, including the drive select and behavior in low-power mode.

Parameters

<i>base</i>	PMC peripheral base address.
<i>config</i>	Pointer to the configuration structure

### 23.5.8 static bool PMC\_GetPeriphIOIsolationFlag ( **PMC\_Type** \* *base* ) [**inline**], [**static**]

This function reads the Acknowledge Isolation setting that indicates whether certain peripherals and the I/O pads are in a latched state as a result of having been in the VLLS mode.

Parameters

<i>base</i>	PMC peripheral base address.
<i>base</i>	Base address for current PMC instance.

Returns

ACK isolation 0 - Peripherals and I/O pads are in a normal run state. 1 - Certain peripherals and I/O pads are in an isolated and latched state.

### 23.5.9 static void PMC\_ClearPeriphIOIsolationFlag ( **PMC\_Type** \* *base* ) [**inline**], [**static**]

This function clears the ACK Isolation flag. Writing one to this setting when it is set releases the I/O pads and certain peripherals to their normal run mode state.

Parameters

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

### 23.5.10 static bool PMC\_IsRegulatorInRunRegulation ( **PMC\_Type** \* *base* ) [**inline**], [**static**]

This function returns the regulator to run a regulation status. It provides the current status of the internal voltage regulator.

## Parameters

<i>base</i>	PMC peripheral base address.
<i>base</i>	Base address for current PMC instance.

## Returns

Regulation status 0 - Regulator is in a stop regulation or in transition to/from the regulation. 1 - Regulator is in a run regulation.

# Chapter 24

## PORT: Port Control and Interrupts

### 24.1 Overview

The MCUXpresso SDK provides a driver for the Port Control and Interrupts (PORT) module of MCUXpresso SDK devices.

### Data Structures

- struct `port_digital_filter_config_t`  
*PORT digital filter feature configuration definition.* [More...](#)
- struct `port_pin_config_t`  
*PORT pin configuration structure.* [More...](#)

### Enumerations

- enum `_port_pull` {  
  `kPORT_PullDisable` = 0U,  
  `kPORT_PullDown` = 2U,  
  `kPORT_PullUp` = 3U }  
*Internal resistor pull feature selection.*
- enum `_port_slew_rate` {  
  `kPORT_FastSlewRate` = 0U,  
  `kPORT_SlowSlewRate` = 1U }  
*Slew rate selection.*
- enum `_port_open_drain_enable` {  
  `kPORT_OpenDrainDisable` = 0U,  
  `kPORT_OpenDrainEnable` = 1U }  
*Open Drain feature enable/disable.*
- enum `_port_lock_register` {  
  `kPORT_UnlockRegister` = 0U,  
  `kPORT_LockRegister` = 1U }  
*Unlock/lock the pin control register field[15:0].*
- enum `port_mux_t` {

```
kPORT_PinDisabledOrAnalog = 0U,
kPORT_MuxAsGpio = 1U,
kPORT_MuxAlt2 = 2U,
kPORT_MuxAlt3 = 3U,
kPORT_MuxAlt4 = 4U,
kPORT_MuxAlt5 = 5U,
kPORT_MuxAlt6 = 6U,
kPORT_MuxAlt7 = 7U,
kPORT_MuxAlt8 = 8U,
kPORT_MuxAlt9 = 9U,
kPORT_MuxAlt10 = 10U,
kPORT_MuxAlt11 = 11U,
kPORT_MuxAlt12 = 12U,
kPORT_MuxAlt13 = 13U,
kPORT_MuxAlt14 = 14U,
kPORT_MuxAlt15 = 15U }
```

*Pin mux selection.*

- enum `port_interrupt_t` {
 

```
kPORT_InterruptOrDMADisabled = 0x0U,
kPORT_DMARisingEdge = 0x1U,
kPORT_DMAFallingEdge = 0x2U,
kPORT_DMAEitherEdge = 0x3U,
kPORT_FlagRisingEdge = 0x05U,
kPORT_FlagFallingEdge = 0x06U,
kPORT_FlagEitherEdge = 0x07U,
kPORT_InterruptLogicZero = 0x8U,
kPORT_InterruptRisingEdge = 0x9U,
kPORT_InterruptFallingEdge = 0xAU,
kPORT_InterruptEitherEdge = 0xBU,
kPORT_InterruptLogicOne = 0xCU,
kPORT_ActiveHighTriggerOutputEnable = 0xDU,
kPORT_ActiveLowTriggerOutputEnable = 0xEU }
```

*Configures the interrupt generation condition.*

- enum `port_digital_filter_clock_source_t` {
 

```
kPORT_BusClock = 0U,
kPORT_LpoClock = 1U }
```

*Digital filter clock source selection.*

## Driver version

- #define `FSL_PORT_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 0)`)  
*PORT driver version.*

## Configuration

- static void `PORT_SetPinConfig` (`PORT_Type` \*base, `uint32_t` pin, const `port_pin_config_t` \*config)

- static void [PORT\\_SetMultiplePinsConfig](#) (PORT\_Type \*base, uint32\_t mask, const [port\\_pin\\_config\\_t](#) \*config)
 

*Sets the port PCR register.*
- static void [PORT\\_SetPinMux](#) (PORT\_Type \*base, uint32\_t pin, [port\\_mux\\_t](#) mux)
 

*Sets the port PCR register for multiple pins.*
- static void [PORT\\_EnablePinsDigitalFilter](#) (PORT\_Type \*base, uint32\_t mask, bool enable)
 

*Configures the pin muxing.*
- static void [PORT\\_SetDigitalFilterConfig](#) (PORT\_Type \*base, const [port\\_digital\\_filter\\_config\\_t](#) \*config)
 

*Enables the digital filter in one port, each bit of the 32-bit register represents one pin.*
- static void [PORT\\_SetPinInterruptConfig](#) (PORT\_Type \*base, uint32\_t pin, [port\\_interrupt\\_t](#) config)
 

*Configures the port pin interrupt/DMA request.*
- static uint32\_t [PORT\\_GetPinsInterruptFlags](#) (PORT\_Type \*base)
 

*Reads the whole port status flag.*
- static void [PORT\\_ClearPinsInterruptFlags](#) (PORT\_Type \*base, uint32\_t mask)
 

*Clears the multiple pin interrupt status flag.*

## Interrupt

- static void [PORT\\_SetPinInterruptConfig](#) (PORT\_Type \*base, uint32\_t pin, [port\\_interrupt\\_t](#) config)
 

*Configures the port pin interrupt/DMA request.*
- static uint32\_t [PORT\\_GetPinsInterruptFlags](#) (PORT\_Type \*base)
 

*Reads the whole port status flag.*
- static void [PORT\\_ClearPinsInterruptFlags](#) (PORT\_Type \*base, uint32\_t mask)
 

*Clears the multiple pin interrupt status flag.*

## 24.2 Data Structure Documentation

### 24.2.1 struct port\_digital\_filter\_config\_t

#### Data Fields

- uint32\_t [digitalFilterWidth](#)

*Set digital filter width.*
- [port\\_digital\\_filter\\_clock\\_source\\_t](#) [clockSource](#)

*Set digital filter clockSource.*

### 24.2.2 struct port\_pin\_config\_t

#### Data Fields

- uint16\_t [pullSelect](#): 2
 

*No-pull/pull-down/pull-up select.*
- uint16\_t [slewRate](#): 1
 

*Fast/slow slew rate Configure.*
- uint16\_t [openDrainEnable](#): 1
 

*Open drain enable/disable.*
- uint16\_t [mux](#): 3
 

*Pin mux Configure.*
- uint16\_t [lockRegister](#): 1
 

*Lock/unlock the PCR field[15:0].*

## 24.3 Macro Definition Documentation

### 24.3.1 #define FSL\_PORT\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 0))

## 24.4 Enumeration Type Documentation

### 24.4.1 enum \_port\_pull

Enumerator

*kPORT\_PullDisable* Internal pull-up/down resistor is disabled.

*kPORT\_PullDown* Internal pull-down resistor is enabled.

*kPORT\_PullUp* Internal pull-up resistor is enabled.

### 24.4.2 enum \_port\_slew\_rate

Enumerator

*kPORT\_FastSlewRate* Fast slew rate is configured.

*kPORT\_SlowSlewRate* Slow slew rate is configured.

### 24.4.3 enum \_port\_open\_drain\_enable

Enumerator

*kPORT\_OpenDrainDisable* Open drain output is disabled.

*kPORT\_OpenDrainEnable* Open drain output is enabled.

### 24.4.4 enum \_port\_lock\_register

Enumerator

*kPORT\_UnlockRegister* Pin Control Register fields [15:0] are not locked.

*kPORT\_LockRegister* Pin Control Register fields [15:0] are locked.

### 24.4.5 enum port\_mux\_t

Enumerator

*kPORT\_PinDisabledOrAnalog* Corresponding pin is disabled, but is used as an analog pin.

*kPORT\_MuxAsGpio* Corresponding pin is configured as GPIO.

***kPORT\_MuxAlt2*** Chip-specific.  
***kPORT\_MuxAlt3*** Chip-specific.  
***kPORT\_MuxAlt4*** Chip-specific.  
***kPORT\_MuxAlt5*** Chip-specific.  
***kPORT\_MuxAlt6*** Chip-specific.  
***kPORT\_MuxAlt7*** Chip-specific.  
***kPORT\_MuxAlt8*** Chip-specific.  
***kPORT\_MuxAlt9*** Chip-specific.  
***kPORT\_MuxAlt10*** Chip-specific.  
***kPORT\_MuxAlt11*** Chip-specific.  
***kPORT\_MuxAlt12*** Chip-specific.  
***kPORT\_MuxAlt13*** Chip-specific.  
***kPORT\_MuxAlt14*** Chip-specific.  
***kPORT\_MuxAlt15*** Chip-specific.

#### 24.4.6 enum port\_interrupt\_t

Enumerator

***kPORT\_InterruptOrDMADisabled*** Interrupt/DMA request is disabled.  
***kPORT\_DMARisingEdge*** DMA request on rising edge.  
***kPORT\_DMAFallingEdge*** DMA request on falling edge.  
***kPORT\_DMAEitherEdge*** DMA request on either edge.  
***kPORT\_FlagRisingEdge*** Flag sets on rising edge.  
***kPORT\_FlagFallingEdge*** Flag sets on falling edge.  
***kPORT\_FlagEitherEdge*** Flag sets on either edge.  
***kPORT\_InterruptLogicZero*** Interrupt when logic zero.  
***kPORT\_InterruptRisingEdge*** Interrupt on rising edge.  
***kPORT\_InterruptFallingEdge*** Interrupt on falling edge.  
***kPORT\_InterruptEitherEdge*** Interrupt on either edge.  
***kPORT\_InterruptLogicOne*** Interrupt when logic one.  
***kPORT\_ActiveHighTriggerOutputEnable*** Enable active high-trigger output.  
***kPORT\_ActiveLowTriggerOutputEnable*** Enable active low-trigger output.

#### 24.4.7 enum port\_digital\_filter\_clock\_source\_t

Enumerator

***kPORT\_BusClock*** Digital filters are clocked by the bus clock.  
***kPORT\_LpoClock*** Digital filters are clocked by the 1 kHz LPO clock.

### 24.5 Function Documentation

#### 24.5.1 static void PORT\_SetPinConfig ( PORT\_Type \* *base*, uint32\_t *pin*, const port\_pin\_config\_t \* *config* ) [inline], [static]

This is an example to define an input pin or output pin PCR configuration.

```
* // Define a digital input pin PCR configuration
* port_pin_config_t config = {
*     kPORT_PullUp,
*     kPORT_FastSlewRate,
*     kPORT_PassiveFilterDisable,
*     kPORT_OpenDrainDisable,
*     kPORT_LowDriveStrength,
*     kPORT_MuxAsGpio,
*     kPORT_UnLockRegister,
* };
*
```

Parameters

<i>base</i>	PORT peripheral base pointer.
<i>pin</i>	PORT pin number.
<i>config</i>	PORT PCR register configuration structure.

#### 24.5.2 static void PORT\_SetMultiplePinsConfig ( PORT\_Type \* *base*, uint32\_t *mask*, const port\_pin\_config\_t \* *config* ) [inline], [static]

This is an example to define input pins or output pins PCR configuration.

```
* Define a digital input pin PCR configuration
* port_pin_config_t config = {
*     kPORT_PullUp ,
*     kPORT_PullEnable,
*     kPORT_FastSlewRate,
*     kPORT_PassiveFilterDisable,
*     kPORT_OpenDrainDisable,
*     kPORT_LowDriveStrength,
*     kPORT_MuxAsGpio,
*     kPORT_UnlockRegister,
* };
*
```

Parameters

<i>base</i>	PORT peripheral base pointer.
<i>mask</i>	PORT pin number macro.
<i>config</i>	PORT PCR register configuration structure.

#### 24.5.3 static void PORT\_SetPinMux ( PORT\_Type \* *base*, uint32\_t *pin*, port\_mux\_t *mux* ) [inline], [static]

Parameters

<i>base</i>	PORT peripheral base pointer.
<i>pin</i>	PORT pin number.
<i>mux</i>	<p>pin muxing slot selection.</p> <ul style="list-style-type: none"> <li>• <a href="#">kPORT_PinDisabledOrAnalog</a>: Pin disabled or work in analog function.</li> <li>• <a href="#">kPORT_MuxAsGpio</a> : Set as GPIO.</li> <li>• <a href="#">kPORT_MuxAlt2</a> : chip-specific.</li> <li>• <a href="#">kPORT_MuxAlt3</a> : chip-specific.</li> <li>• <a href="#">kPORT_MuxAlt4</a> : chip-specific.</li> <li>• <a href="#">kPORT_MuxAlt5</a> : chip-specific.</li> <li>• <a href="#">kPORT_MuxAlt6</a> : chip-specific.</li> <li>• <a href="#">kPORT_MuxAlt7</a> : chip-specific.</li> </ul>

Note

: This function is NOT recommended to use together with the PORT\_SetPinsConfig, because the PORT\_SetPinsConfig need to configure the pin mux anyway (Otherwise the pin mux is reset to zero : [kPORT\\_PinDisabledOrAnalog](#)). This function is recommended to use to reset the pin mux

#### 24.5.4 static void PORT\_EnablePinsDigitalFilter ( PORT\_Type \* *base*, uint32\_t *mask*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	PORT peripheral base pointer.
-------------	-------------------------------

<i>mask</i>	PORT pin number macro.
<i>enable</i>	PORT digital filter configuration.

#### 24.5.5 static void PORT\_SetDigitalFilterConfig ( PORT\_Type \* *base*, const port\_digital\_filter\_config\_t \* *config* ) [inline], [static]

Parameters

<i>base</i>	PORT peripheral base pointer.
<i>config</i>	PORT digital filter configuration structure.

#### 24.5.6 static void PORT\_SetPinInterruptConfig ( PORT\_Type \* *base*, uint32\_t *pin*, port\_interrupt\_t *config* ) [inline], [static]

Parameters

<i>base</i>	PORT peripheral base pointer.
<i>pin</i>	PORT pin number.
<i>config</i>	PORT pin interrupt configuration. <ul style="list-style-type: none"> <li>• <a href="#">kPORT_InterruptOrDMADisabled</a>: Interrupt/DMA request disabled.</li> <li>• <a href="#">kPORT_DMARisingEdge</a> : DMA request on rising edge(if the DMA requests exit).</li> <li>• <a href="#">kPORT_DMAPFallingEdge</a>: DMA request on falling edge(if the DMA requests exit).</li> <li>• <a href="#">kPORT_DMAEitherEdge</a> : DMA request on either edge(if the DMA requests exit).</li> <li>• <a href="#">kPORT_FlagRisingEdge</a> : Flag sets on rising edge(if the Flag states exit).</li> <li>• <a href="#">kPORT_FlagFallingEdge</a> : Flag sets on falling edge(if the Flag states exit).</li> <li>• <a href="#">kPORT_FlagEitherEdge</a> : Flag sets on either edge(if the Flag states exit).</li> <li>• <a href="#">kPORT_InterruptLogicZero</a> : Interrupt when logic zero.</li> <li>• <a href="#">kPORT_InterruptRisingEdge</a> : Interrupt on rising edge.</li> <li>• <a href="#">kPORT_InterruptFallingEdge</a>: Interrupt on falling edge.</li> <li>• <a href="#">kPORT_InterruptEitherEdge</a> : Interrupt on either edge.</li> <li>• <a href="#">kPORT_InterruptLogicOne</a> : Interrupt when logic one.</li> <li>• <a href="#">kPORT_ActiveHighTriggerOutputEnable</a> : Enable active high-trigger output (if the trigger states exit).</li> <li>• <a href="#">kPORT_ActiveLowTriggerOutputEnable</a> : Enable active low-trigger output (if the trigger states exit).</li> </ul>

**24.5.7 static uint32\_t PORT\_GetPinsInterruptFlags ( PORT\_Type \* *base* )  
[inline], [static]**

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

<i>base</i>	PORT peripheral base pointer.
-------------	-------------------------------

Returns

Current port interrupt status flags, for example, 0x00010001 means the pin 0 and 16 have the interrupt.

**24.5.8 static void PORT\_ClearPinsInterruptFlags ( PORT\_Type \* *base*, uint32\_t  
*mask* ) [inline], [static]**

Parameters

<i>base</i>	PORT peripheral base pointer.
<i>mask</i>	PORT pin number macro.

# Chapter 25

## QTMR: Quad Timer Driver

### 25.1 Overview

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

### Data Structures

- struct `qtmr_config_t`  
*Quad Timer config structure. [More...](#)*

### Enumerations

- enum `qtmr_primary_count_source_t` {  
  `kQTMR_ClockCounter0InputPin` = 0,  
  `kQTMR_ClockCounter1InputPin`,  
  `kQTMR_ClockCounter2InputPin`,  
  `kQTMR_ClockCounter3InputPin`,  
  `kQTMR_ClockCounter0Output`,  
  `kQTMR_ClockCounter1Output`,  
  `kQTMR_ClockCounter2Output`,  
  `kQTMR_ClockCounter3Output`,  
  `kQTMR_ClockDivide_1`,  
  `kQTMR_ClockDivide_2`,  
  `kQTMR_ClockDivide_4`,  
  `kQTMR_ClockDivide_8`,  
  `kQTMR_ClockDivide_16`,  
  `kQTMR_ClockDivide_32`,  
  `kQTMR_ClockDivide_64`,  
  `kQTMR_ClockDivide_128` }  
*Quad Timer primary clock source selection.*
- enum `qtmr_input_source_t` {  
  `kQTMR_Counter0InputPin` = 0,  
  `kQTMR_Counter1InputPin`,  
  `kQTMR_Counter2InputPin`,  
  `kQTMR_Counter3InputPin` }  
*Quad Timer input sources selection.*
- enum `qtmr_counting_mode_t` {

```
kQTMR_NoOperation = 0,
kQTMR_PriSrcRiseEdge,
kQTMR_PriSrcRiseAndFallEdge,
kQTMR_PriSrcRiseEdgeSecInpHigh,
kQTMR_QuadCountMode,
kQTMR_PriSrcRiseEdgeSecDir,
kQTMR_SecSrcTrigPriCnt,
kQTMR_CascadeCount }
```

*Quad Timer counting mode selection.*

- enum `qtmr_output_mode_t` {
   
kQTMR\_AssertWhenCountActive = 0,
   
kQTMR\_ClearOnCompare,
   
kQTMR\_SetOnCompare,
   
kQTMR\_ToggleOnCompare,
   
kQTMR\_ToggleOnAltCompareReg,
   
kQTMR\_SetOnCompareClearOnSecSrcInp,
   
kQTMR\_SetOnCompareClearOnCountRoll,
   
kQTMR\_EnableGateClock }

*Quad Timer output mode selection.*

- enum `qtmr_input_capture_edge_t` {
   
kQTMR\_NoCapture = 0,
   
kQTMR\_RisingEdge,
   
kQTMR\_FallingEdge,
   
kQTMR\_RisingAndFallingEdge }

*Quad Timer input capture edge mode, rising edge, or falling edge.*

- enum `qtmr_reload_control_t` {
   
kQTMR\_NoPreload = 0,
   
kQTMR\_LoadOnComp1,
   
kQTMR\_LoadOnComp2 }

*Quad Timer input capture edge mode, rising edge, or falling edge.*

- enum `qtmr_debug_action_t` {
   
kQTMR\_RunNormalInDebug = 0U,
   
kQTMR\_HaltCounter,
   
kQTMR\_ForceOutToZero,
   
kQTMR\_HaltCountForceOutZero }

*List of Quad Timer run options when in Debug mode.*

- enum `qtmr_interrupt_enable_t` {
   
kQTMR\_CompareInterruptEnable = (1U << 0),
   
kQTMR\_Compare1InterruptEnable = (1U << 1),
   
kQTMR\_Compare2InterruptEnable = (1U << 2),
   
kQTMR\_OverflowInterruptEnable = (1U << 3),
   
kQTMR\_EdgeInterruptEnable = (1U << 4) }

*List of Quad Timer interrupts.*

- enum `qtmr_status_flags_t` {

```
kQTMR_CompareFlag = (1U << 0),
kQTMR_Compare1Flag = (1U << 1),
kQTMR_Compare2Flag = (1U << 2),
kQTMR_OverflowFlag = (1U << 3),
kQTMR_EdgeFlag = (1U << 4) }
```

*List of Quad Timer flags.*

## Functions

- `status_t QTMR_SetupPwm (TMR_Type *base, uint32_t pwmFreqHz, uint8_t dutyCyclePercent, bool outputPolarity, uint32_t srcClock_Hz)`

*Sets up Quad timer module for PWM signal output.*
- `void QTMR_SetupInputCapture (TMR_Type *base, qtmr_input_source_t capturePin, bool inputPolarity, bool reloadOnCapture, qtmr_input_capture_edge_t captureMode)`

*Allows the user to count the source clock cycles until a capture event arrives.*

## Driver version

- `#define FSL_QTMR_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))`

*Version.*

## Initialization and deinitialization

- `void QTMR_Init (TMR_Type *base, const qtmr_config_t *config)`

*Ungates the Quad Timer clock and configures the peripheral for basic operation.*
- `void QTMR_Deinit (TMR_Type *base)`

*Stops the counter and gates the Quad Timer clock.*
- `void QTMR_GetDefaultConfig (qtmr_config_t *config)`

*Fill in the Quad Timer config struct with the default settings.*

## Interrupt Interface

- `void QTMR_EnableInterrupts (TMR_Type *base, uint32_t mask)`

*Enables the selected Quad Timer interrupts.*
- `void QTMR_DisableInterrupts (TMR_Type *base, uint32_t mask)`

*Disables the selected Quad Timer interrupts.*
- `uint32_t QTMR_GetEnabledInterrupts (TMR_Type *base)`

*Gets the enabled Quad Timer interrupts.*

## Status Interface

- `uint32_t QTMR_GetStatus (TMR_Type *base)`

*Gets the Quad Timer status flags.*
- `void QTMR_ClearStatusFlags (TMR_Type *base, uint32_t mask)`

*Clears the Quad Timer status flags.*

## Read and Write the timer period

- `void QTMR_SetTimerPeriod (TMR_Type *base, uint16_t ticks)`

- Sets the timer period in ticks.
- static uint16\_t [QTMR\\_GetCurrentTimerCount](#) (TMR\_Type \*base)  
Reads the current timer counting value.

## Timer Start and Stop

- static void [QTMR\\_StartTimer](#) (TMR\_Type \*base, [qtmr\\_counting\\_mode\\_t](#) clockSource)  
Starts the Quad Timer counter.
- static void [QTMR\\_StopTimer](#) (TMR\_Type \*base)  
Stops the Quad Timer counter.

## 25.2 Data Structure Documentation

### 25.2.1 struct qtmr\_config\_t

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

The config struct can be made const so it resides in flash

### Data Fields

- [qtmr\\_primary\\_count\\_source\\_t](#) primarySource  
Specify the primary count source.
- [qtmr\\_input\\_source\\_t](#) secondarySource  
Specify the secondary count source.
- bool enableMasterMode  
*true: Broadcast compare function output to other counters; false no broadcast*
- bool enableExternalForce  
*true: Compare from another counter force state of OFLAG signal false: OFLAG controlled by local counter*
- uint8\_t faultFilterCount  
Fault filter count.
- uint8\_t faultFilterPeriod  
Fault filter period; value of 0 will bypass the filter.
- [qtmr\\_debug\\_action\\_t](#) debugMode  
Operation in Debug mode.

## 25.3 Macro Definition Documentation

### 25.3.1 #define FSL\_QTMR\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

## 25.4 Enumeration Type Documentation

### 25.4.1 enum qtmr\_primary\_count\_source\_t

Enumerator

- kQTMR\_ClockCounter0InputPin* Use counter 0 input pin.
- kQTMR\_ClockCounter1InputPin* Use counter 1 input pin.
- kQTMR\_ClockCounter2InputPin* Use counter 2 input pin.
- kQTMR\_ClockCounter3InputPin* Use counter 3 input pin.
- kQTMR\_ClockCounter0Output* Use counter 0 output.
- kQTMR\_ClockCounter1Output* Use counter 1 output.
- kQTMR\_ClockCounter2Output* Use counter 2 output.
- kQTMR\_ClockCounter3Output* Use counter 3 output.
- kQTMR\_ClockDivide\_1* IP bus clock divide by 1 prescaler.
- kQTMR\_ClockDivide\_2* IP bus clock divide by 2 prescaler.
- kQTMR\_ClockDivide\_4* IP bus clock divide by 4 prescaler.
- kQTMR\_ClockDivide\_8* IP bus clock divide by 8 prescaler.
- kQTMR\_ClockDivide\_16* IP bus clock divide by 16 prescaler.
- kQTMR\_ClockDivide\_32* IP bus clock divide by 32 prescaler.
- kQTMR\_ClockDivide\_64* IP bus clock divide by 64 prescaler.
- kQTMR\_ClockDivide\_128* IP bus clock divide by 128 prescaler.

### 25.4.2 enum qtmr\_input\_source\_t

Enumerator

- kQTMR\_Counter0InputPin* Use counter 0 input pin.
- kQTMR\_Counter1InputPin* Use counter 1 input pin.
- kQTMR\_Counter2InputPin* Use counter 2 input pin.
- kQTMR\_Counter3InputPin* Use counter 3 input pin.

### 25.4.3 enum qtmr\_counting\_mode\_t

Enumerator

- kQTMR\_NoOperation* No operation.
- kQTMR\_PriSrcRiseEdge* Count rising edges of primary source.
- kQTMR\_PriSrcRiseAndFallEdge* Count rising and falling edges of primary source.
- kQTMR\_PriSrcRiseEdgeSecInpHigh* Count rise edges of pri SRC while sec inp high active.
- kQTMR\_QuadCountMode* Quadrature count mode, uses pri and sec sources.
- kQTMR\_PriSrcRiseEdgeSecDir* Count rising edges of pri SRC; sec SRC specifies dir.
- kQTMR\_SecSrcTrigPriCnt* Edge of sec SRC trigger primary count until compare.
- kQTMR\_CascadeCount* Cascaded count mode (up/down)

#### 25.4.4 enum qtmr\_output\_mode\_t

Enumerator

- kQTMR AssertWhenCountActive* Assert OFLAG while counter is active.
- kQTMR ClearOnCompare* Clear OFLAG on successful compare.
- kQTMR SetOnCompare* Set OFLAG on successful compare.
- kQTMR ToggleOnCompare* Toggle OFLAG on successful compare.
- kQTMR ToggleOnAltCompareReg* Toggle OFLAG using alternating compare registers.
- kQTMR SetOnCompareClearOnSecSrcInp* Set OFLAG on compare, clear on sec SRC input edge.
  
- kQTMR SetOnCompareClearOnCountRoll* Set OFLAG on compare, clear on counter rollover.
- kQTMR EnableGateClock* Enable gated clock output while count is active.

#### 25.4.5 enum qtmr\_input\_capture\_edge\_t

Enumerator

- kQTMR NoCapture* Capture is disabled.
- kQTMR RisingEdge* Capture on rising edge (IPS=0) or falling edge (IPS=1)
- kQTMR FallingEdge* Capture on falling edge (IPS=0) or rising edge (IPS=1)
- kQTMR RisingAndFallingEdge* Capture on both edges.

#### 25.4.6 enum qtmr\_reload\_control\_t

Enumerator

- kQTMR NoPreload* Never preload.
- kQTMR LoadOnComp1* Load upon successful compare with value in COMP1.
- kQTMR LoadOnComp2* Load upon successful compare with value in COMP2.

#### 25.4.7 enum qtmr\_debug\_action\_t

Enumerator

- kQTMR RunNormalInDebug* Continue with normal operation.
- kQTMR HaltCounter* Halt counter.
- kQTMR ForceOutToZero* Force output to logic 0.
- kQTMR HaltCountForceOutZero* Halt counter and force output to logic 0.

### 25.4.8 enum qtmr\_interrupt\_enable\_t

Enumerator

*kQTMR\_CompareInterruptEnable* Compare interrupt.  
*kQTMR\_Compare1InterruptEnable* Compare 1 interrupt.  
*kQTMR\_Compare2InterruptEnable* Compare 2 interrupt.  
*kQTMR\_OverflowInterruptEnable* Timer overflow interrupt.  
*kQTMR\_EdgeInterruptEnable* Input edge interrupt.

### 25.4.9 enum qtmr\_status\_flags\_t

Enumerator

*kQTMR\_CompareFlag* Compare flag.  
*kQTMR\_Compare1Flag* Compare 1 flag.  
*kQTMR\_Compare2Flag* Compare 2 flag.  
*kQTMR\_OverflowFlag* Timer overflow flag.  
*kQTMR\_EdgeFlag* Input edge flag.

## 25.5 Function Documentation

### 25.5.1 void QTMR\_Init ( TMR\_Type \* *base*, const qtmr\_config\_t \* *config* )

Note

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

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>config</i>	Pointer to user's Quad Timer config structure

### 25.5.2 void QTMR\_Deinit ( TMR\_Type \* *base* )

Parameters

<i>base</i>	Quad Timer peripheral base address
-------------	------------------------------------

### 25.5.3 void QTMR\_GetDefaultConfig ( qtmr\_config\_t \* *config* )

The default values are:

```
* config->debugMode = kQTMR_RunNormalInDebug;
* config->enableExternalForce = false;
* config->enableMasterMode = false;
* config->faultFilterCount = 0;
* config->faultFilterPeriod = 0;
* config->primarySource = kQTMR_ClockDivide_2;
* config->secondarySource = kQTMR_Counter0InputPin;
*
```

Parameters

<i>config</i>	Pointer to user's Quad Timer config structure.
---------------	--

### 25.5.4 status\_t QTMR\_SetupPwm ( TMR\_Type \* *base*, uint32\_t *pwmFreqHz*, uint8\_t *dutyCyclePercent*, bool *outputPolarity*, uint32\_t *srcClock\_Hz* )

The function initializes the timer module according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements.

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>pwmFreqHz</i>	PWM signal frequency in Hz
<i>dutyCyclePercent</i>	PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)
<i>outputPolarity</i>	true: invert polarity of the output signal, false: no inversion
<i>srcClock_Hz</i>	Main counter clock in Hz.

Returns

Returns an error if there was error setting up the signal.

25.5.5 **void QTMR\_SetupInputCapture ( TMR\_Type \* *base*, qtmr\_input\_source\_t *capturePin*, bool *inputPolarity*, bool *reloadOnCapture*, qtmr\_input\_capture\_edge\_t *captureMode* )**

The count is stored in the capture register.

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>capturePin</i>	Pin through which we receive the input signal to trigger the capture
<i>inputPolarity</i>	true: invert polarity of the input signal, false: no inversion
<i>reloadOn-Capture</i>	true: reload the counter when an input capture occurs, false: no reload
<i>captureMode</i>	Specifies which edge of the input signal triggers a capture

### 25.5.6 void QTMR\_EnableInterrupts ( TMR\_Type \* *base*, uint32\_t *mask* )

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">qtmr_interrupt_enable_t</a>

### 25.5.7 void QTMR\_DisableInterrupts ( TMR\_Type \* *base*, uint32\_t *mask* )

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">qtmr_interrupt_enable_t</a>

### 25.5.8 uint32\_t QTMR\_GetEnabledInterrupts ( TMR\_Type \* *base* )

Parameters

<i>base</i>	Quad Timer peripheral base address
-------------	------------------------------------

Returns

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

25.5.9 `uint32_t QTMR_GetStatus ( TMR_Type * base )`

Parameters

<i>base</i>	Quad Timer peripheral base address
-------------	------------------------------------

Returns

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

### 25.5.10 void QTMR\_ClearStatusFlags ( TMR\_Type \* *base*, uint32\_t *mask* )

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>mask</i>	The status flags to clear. This is a logical OR of members of the enumeration <a href="#">qtmr_status_flags_t</a>

### 25.5.11 void QTMR\_SetTimerPeriod ( TMR\_Type \* *base*, uint16\_t *ticks* )

Timers counts from initial value till it equals the count value set here. The counter will then reinitialize to the value specified in the Load register.

Note

1. This function will write the time period in ticks to COMP1 or COMP2 register depending on the count direction
2. User can call the utility macros provided in fsl\_common.h to convert to ticks
3. This function supports cases, providing only primary source clock without secondary source clock.

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>ticks</i>	Timer period in units of ticks

### 25.5.12 static uint16\_t QTMR\_GetCurrentTimerCount ( TMR\_Type \* *base* ) [inline], [static]

This function returns the real-time timer counting value, in a range from 0 to a timer period.

## Note

User can call the utility macros provided in fsl\_common.h to convert ticks to usec or msec

## Parameters

<i>base</i>	Quad Timer peripheral base address
-------------	------------------------------------

## Returns

Current counter value in ticks

**25.5.13 static void QTMR\_StartTimer( TMR\_Type \* *base*, qtmr\_counting\_mode\_t *clockSource* ) [inline], [static]**

## Parameters

<i>base</i>	Quad Timer peripheral base address
<i>clockSource</i>	Quad Timer clock source

**25.5.14 static void QTMR\_StopTimer( TMR\_Type \* *base* ) [inline], [static]**

## Parameters

<i>base</i>	Quad Timer peripheral base address
-------------	------------------------------------

# Chapter 26

## RCM: Reset Control Module Driver

### 26.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Reset Control Module (RCM) module of MCUXpresso SDK devices.

### Data Structures

- struct `rcm_reset_pin_filter_config_t`  
*Reset pin filter configuration. [More...](#)*

### Enumerations

- enum `rcm_reset_source_t` {  
    `kRCM_SourceWakeup` = RCM\_SRS0\_WAKEUP\_MASK,  
    `kRCM_SourceLvd` = RCM\_SRS0\_LVD\_MASK,  
    `kRCM_SourceLoc` = RCM\_SRS0\_LOC\_MASK,  
    `kRCM_SourceLol` = RCM\_SRS0\_LOL\_MASK,  
    `kRCM_SourceWdog` = RCM\_SRS0\_WDOG\_MASK,  
    `kRCM_SourcePin` = RCM\_SRS0\_PIN\_MASK,  
    `kRCM_SourcePor` = RCM\_SRS0\_POR\_MASK,  
    `kRCM_SourceLockup` = RCM\_SRS1\_LOCKUP\_MASK << 8U,  
    `kRCM_SourceSw` = RCM\_SRS1\_SW\_MASK << 8U,  
    `kRCM_SourceMdmap` = RCM\_SRS1\_MDM\_AP\_MASK << 8U,  
    `kRCM_SourceSackerr` = RCM\_SRS1\_SACKERR\_MASK << 8U }  
    *System Reset Source Name definitions.*
- enum `rcm_run_wait_filter_mode_t` {  
    `kRCM_FilterDisable` = 0U,  
    `kRCM_FilterBusClock` = 1U,  
    `kRCM_FilterLpoClock` = 2U }  
    *Reset pin filter select in Run and Wait modes.*

### Driver version

- #define `FSL_RCM_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 4)`)  
*RCM driver version 2.0.4.*

### Reset Control Module APIs

- static `uint32_t RCM_GetPreviousResetSources` (`RCM_Type *base`)  
*Gets the reset source status which caused a previous reset.*
- static `uint32_t RCM_GetStickyResetSources` (`RCM_Type *base`)

- Gets the sticky reset source status.
- static void [RCM\\_ClearStickyResetSources](#) (RCM\_Type \*base, uint32\_t sourceMasks)  
Clears the sticky reset source status.
- void [RCM\\_ConfigureResetPinFilter](#) (RCM\_Type \*base, const [rcm\\_reset\\_pin\\_filter\\_config\\_t](#) \*config)  
Configures the reset pin filter.

## 26.2 Data Structure Documentation

### 26.2.1 struct rcm\_reset\_pin\_filter\_config\_t

#### Data Fields

- bool [enableFilterInStop](#)  
Reset pin filter select in stop mode.
- [rcm\\_run\\_wait\\_filter\\_mode\\_t](#) [filterInRunWait](#)  
Reset pin filter in run/wait mode.
- uint8\_t [busClockFilterCount](#)  
Reset pin bus clock filter width.

#### Field Documentation

- (1) bool [rcm\\_reset\\_pin\\_filter\\_config\\_t::enableFilterInStop](#)
- (2) [rcm\\_run\\_wait\\_filter\\_mode\\_t](#) [rcm\\_reset\\_pin\\_filter\\_config\\_t::filterInRunWait](#)
- (3) uint8\_t [rcm\\_reset\\_pin\\_filter\\_config\\_t::busClockFilterCount](#)

## 26.3 Macro Definition Documentation

### 26.3.1 #define FSL\_RCM\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 4))

## 26.4 Enumeration Type Documentation

### 26.4.1 enum rcm\_reset\_source\_t

Enumerator

*kRCM\_SourceWakeup* Low-leakage wakeup reset.

*kRCM\_SourceLvd* Low-voltage detect reset.

*kRCM\_SourceLoc* Loss of clock reset.

*kRCM\_SourceLol* Loss of lock reset.

*kRCM\_SourceWdog* Watchdog reset.

*kRCM\_SourcePin* External pin reset.

*kRCM\_SourcePor* Power on reset.

*kRCM\_SourceLockup* Core lock up reset.

*kRCM\_SourceSw* Software reset.

*kRCM\_SourceMdmap* MDM-AP system reset.

*kRCM\_SourceSackerr* Parameter could get all reset flags.

## 26.4.2 enum rcm\_run\_wait\_filter\_mode\_t

Enumerator

- kRCM\_FilterDisable* All filtering disabled.
- kRCM\_FilterBusClock* Bus clock filter enabled.
- kRCM\_FilterLpoClock* LPO clock filter enabled.

## 26.5 Function Documentation

### 26.5.1 static uint32\_t RCM\_GetPreviousResetSources ( RCM\_Type \* *base* ) [inline], [static]

This function gets the current reset source status. Use source masks defined in the rcm\_reset\_source\_t to get the desired source status.

This is an example.

```
* uint32_t resetStatus;
*
* To get all reset source statuses.
* resetStatus = RCM_GetPreviousResetSources(RCM) & kRCM_SourceAll;
*
* To test whether the MCU is reset using Watchdog.
* resetStatus = RCM_GetPreviousResetSources(RCM) &
    kRCM_SourceWdog;
*
* To test multiple reset sources.
* resetStatus = RCM_GetPreviousResetSources(RCM) & (
    kRCM_SourceWdog | kRCM_SourcePin);
*
```

Parameters

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

Returns

All reset source status bit map.

### 26.5.2 static uint32\_t RCM\_GetStickyResetSources ( RCM\_Type \* *base* ) [inline], [static]

This function gets the current reset source status that has not been cleared by software for a specific source.

This is an example.

```
* uint32_t resetStatus;
*
```

```

* To get all reset source statuses.
* resetStatus = RCM_GetStickyResetSources(RCM) & kRCM_SourceAll;
*
* To test whether the MCU is reset using Watchdog.
* resetStatus = RCM_GetStickyResetSources(RCM) &
    kRCM_SourceWdog;
*
* To test multiple reset sources.
* resetStatus = RCM_GetStickyResetSources(RCM) &
    (kRCM_SourceWdog | kRCM_SourcePin);
*

```

## Parameters

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

## Returns

All reset source status bit map.

### 26.5.3 static void RCM\_ClearStickyResetSources ( RCM\_Type \* *base*, uint32\_t *sourceMasks* ) [inline], [static]

This function clears the sticky system reset flags indicated by source masks.

This is an example.

```

* Clears multiple reset sources.
* RCM_ClearStickyResetSources(kRCM_SourceWdog |
    kRCM_SourcePin);
*

```

## Parameters

<i>base</i>	RCM peripheral base address.
<i>sourceMasks</i>	reset source status bit map

### 26.5.4 void RCM\_ConfigureResetPinFilter ( RCM\_Type \* *base*, const rcm\_reset\_pin\_filter\_config\_t \* *config* )

This function sets the reset pin filter including the filter source, filter width, and so on.

## Parameters

<i>base</i>	RCM peripheral base address.
<i>config</i>	Pointer to the configuration structure.

# Chapter 27

## RNGA: Random Number Generator Accelerator Driver

### 27.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Random Number Generator Accelerator (RNGA) block of MCUXpresso SDK devices.

### 27.2 RNGA Initialization

1. To initialize the RNGA module, call the [RNGA\\_Init\(\)](#) function. This function automatically enables the RNGA module and its clock.
2. After calling the [RNGA\\_Init\(\)](#) function, the RNGA is enabled and the counter starts working.
3. To disable the RNGA module, call the [RNGA\\_Deinit\(\)](#) function.

### 27.3 Get random data from RNGA

1. [RNGA\\_GetRandomData\(\)](#) function gets random data from the RNGA module.

### 27.4 RNGA Set/Get Working Mode

The RNGA works either in sleep mode or normal mode

1. [RNGA\\_SetMode\(\)](#) function sets the RNGA mode.
2. [RNGA\\_GetMode\(\)](#) function gets the RNGA working mode.

### 27.5 Seed RNGA

1. [RNGA\\_Seed\(\)](#) function inputs an entropy value that the RNGA can use to seed the pseudo random algorithm.

This example code shows how to initialize and get random data from the RNGA driver:

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

#### Note

It is important to note that there is no known cryptographic proof showing this is a secure method for generating random data. In fact, there may be an attack against this random number generator if its output is used directly in a cryptographic application. The attack is based on the linearity of the internal shift registers. Therefore, it is highly recommended that the random data produced by this module be used as an entropy source to provide an input seed to a NIST-approved pseudo-random-number generator based on DES or SHA-1 and defined in NIST FIPS PUB 186-2 Appendix 3 and NIST FIPS PUB SP 800-90. The requirement is needed to maximize the entropy of this input seed. To do this, when data is extracted from RNGA as quickly as the hardware allows, there are one to two bits of added entropy per 32-bit word. Any single bit of that word contains that entropy.

Therefore, when used as an entropy source, a random number should be generated for each bit of entropy required and the least significant bit (any bit would be equivalent) of each word retained. The remainder of each random number should then be discarded. Used this way, even with full knowledge of the internal state of RNGA and all prior random numbers, an attacker is not able to predict the values of the extracted bits. Other sources of entropy can be used along with RNGA to generate the seed to the pseudorandom algorithm. The more random sources combined to create the seed, the better. The following is a list of sources that can be easily combined with the output of this module.

- Current time using highest precision possible
- Real-time system inputs that can be characterized as "random"
- Other entropy supplied directly by the user

## Enumerations

- enum `rnga_mode_t` {
   
    `kRNGA_ModeNormal` = 0U,
   
    `kRNGA_ModeSleep` = 1U
 }
- RNGA working mode.*

## Functions

- void `RNGA_Init` (RNG\_Type \*base)  
*Initializes the RNGA.*
- void `RNGA_Deinit` (RNG\_Type \*base)  
*Shuts down the RNGA.*
- status\_t `RNGA_GetRandomData` (RNG\_Type \*base, void \*data, size\_t data\_size)  
*Gets random data.*
- void `RNGA_Seed` (RNG\_Type \*base, uint32\_t seed)  
*Feeds the RNGA module.*
- void `RNGA_SetMode` (RNG\_Type \*base, `rnga_mode_t` mode)  
*Sets the RNGA in normal mode or sleep mode.*
- `rnga_mode_t RNGA_GetMode` (RNG\_Type \*base)  
*Gets the RNGA working mode.*

## Driver version

- #define `FSL_RNGA_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 2))  
*RNGA driver version 2.0.2.*

## 27.6 Macro Definition Documentation

### 27.6.1 #define FSL\_RNGA\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))

## 27.7 Enumeration Type Documentation

### 27.7.1 enum rnga\_mode\_t

Enumerator

**kRNGA\_ModeNormal** Normal Mode. The ring-oscillator clocks are active; RNGA generates entropy (randomness) from the clocks and stores it in shift registers.

**kRNGA\_ModeSleep** Sleep Mode. The ring-oscillator clocks are inactive; RNGA does not generate entropy.

## 27.8 Function Documentation

### 27.8.1 void RNGA\_Init ( RNG\_Type \* *base* )

This function initializes the RNGA. When called, the RNGA entropy generation starts immediately.

Parameters

<i>base</i>	RNGA base address
-------------	-------------------

### 27.8.2 void RNGA\_Deinit ( RNG\_Type \* *base* )

This function shuts down the RNGA.

Parameters

<i>base</i>	RNGA base address
-------------	-------------------

### 27.8.3 status\_t RNGA\_GetRandomData ( RNG\_Type \* *base*, void \* *data*, size\_t *data\_size* )

This function gets random data from the RNGA.

Parameters

<i>base</i>	RNGA base address
<i>data</i>	pointer to user buffer to be filled by random data
<i>data_size</i>	size of data in bytes

Returns

RNGA status

**27.8.4 void RNGA\_Seed ( RNG\_Type \* *base*, uint32\_t *seed* )**

This function inputs an entropy value that the RNGA uses to seed its pseudo-random algorithm.

Parameters

<i>base</i>	RNGA base address
<i>seed</i>	input seed value

### 27.8.5 void RNGA\_SetMode ( RNG\_Type \* *base*, rnga\_mode\_t *mode* )

This function sets the RNGA in sleep mode or normal mode.

Parameters

<i>base</i>	RNGA base address
<i>mode</i>	normal mode or sleep mode

### 27.8.6 rnga\_mode\_t RNGA\_GetMode ( RNG\_Type \* *base* )

This function gets the RNGA working mode.

Parameters

<i>base</i>	RNGA base address
-------------	-------------------

Returns

normal mode or sleep mode

# Chapter 28

## SIM: System Integration Module Driver

### 28.1 Overview

The MCUXpresso SDK provides a peripheral driver for the System Integration Module (SIM) of MCUXpresso SDK devices.

### Data Structures

- struct `sim_uid_t`  
*Unique ID.* [More...](#)

### Enumerations

- enum `_sim_flash_mode` {  
  `kSIM_FlashDisableInWait` = SIM\_FCFG1\_FLASHDOZE\_MASK,  
  `kSIM_FlashDisable` = SIM\_FCFG1\_FLASHDIS\_MASK }  
*Flash enable mode.*

### Functions

- void `SIM_GetUniqueId` (`sim_uid_t *uid`)  
*Gets the unique identification register value.*
- static void `SIM_SetFlashMode` (`uint8_t mode`)  
*Sets the flash enable mode.*

### Driver version

- #define `FSL_SIM_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 3)`)

### 28.2 Data Structure Documentation

#### 28.2.1 struct `sim_uid_t`

### Data Fields

- `uint32_t H`  
*UIDH.*
- `uint32_t MH`  
*UIDMH.*
- `uint32_t ML`  
*UIDML.*
- `uint32_t L`  
*UIDL.*

## Field Documentation

- (1) `uint32_t sim_uid_t::H`
- (2) `uint32_t sim_uid_t::MH`
- (3) `uint32_t sim_uid_t::ML`
- (4) `uint32_t sim_uid_t::L`

## 28.3 Enumeration Type Documentation

### 28.3.1 `enum _sim_flash_mode`

Enumerator

`kSIM_FlashDisableInWait` Disable flash in wait mode.

`kSIM_FlashDisable` Disable flash in normal mode.

## 28.4 Function Documentation

### 28.4.1 `void SIM_GetUniqueId ( sim_uid_t * uid )`

Parameters

<code>uid</code>	Pointer to the structure to save the UID value.
------------------	---

### 28.4.2 `static void SIM_SetFlashMode ( uint8_t mode ) [inline], [static]`

Parameters

<code>mode</code>	The mode to set; see <a href="#">_sim_flash_mode</a> for mode details.
-------------------	--

# Chapter 29

## SLCD: Segment LCD Driver

### 29.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Segment LCD (SLCD) module of MCUXpresso SDK devices. The SLCD module is a CMOS charge pump voltage inverter that is designed for low voltage and low-power operation. SLCD is designed to generate the appropriate waveforms to drive multiplexed numeric, alphanumeric, or custom segment LCD panels. SLCD also has several timing and control settings that can be software-configured depending on the application's requirements. Timing and control consists of registers and control logic for the following:

1. LCD frame frequency
2. Duty cycle selection
3. Front plane/back plane selection and enabling
4. Blink modes and frequency
5. Operation in low-power modes

### 29.2 Plane Setting and Display Control

After the SLCD general initialization, the [SLCD\\_SetBackPlanePhase\(\)](#), [SLCD\\_SetFrontPlaneSegments\(\)](#), and [SLCD\\_SetFrontPlaneOnePhase\(\)](#) are used to set the special back/front Plane to make SLCD display correctly. Then, the independent display control APIs, [SLCD\\_StartDisplay\(\)](#) and [SLCD\\_StopDisplay\(\)](#), start and stop the SLCD display.

The [SLCD\\_StartBlinkMode\(\)](#) and [SLCD\\_StopBlinkMode\(\)](#) are provided for the runtime special blink mode control. To get the SLCD fault detection result, call the [SLCD\\_GetFaultDetectCounter\(\)](#).

### 29.3 Typical use case

#### 29.3.1 SLCD Initialization operation

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

### Data Structures

- struct [slcd\\_fault\\_detect\\_config\\_t](#)  
*SLCD fault frame detection configuration structure. [More...](#)*
- struct [slcd\\_clock\\_config\\_t](#)  
*SLCD clock configuration structure. [More...](#)*
- struct [slcd\\_config\\_t](#)  
*SLCD configuration structure. [More...](#)*

## Enumerations

- enum `slcd_power_supply_option_t` {
   
kSLCD\_InternalVll3UseChargePump,
   
kSLCD\_ExternalVll3UseResistorBiasNetwork,
   
kSLCD\_ExteranVll3UseChargePump,
   
kSLCD\_InternalVll1UseChargePump }  
*SLCD power supply option.*
- enum `slcd_regulated_voltage_trim_t` {
   
kSLCD\_RegulatedVolatgeTrim00 = 0U,
   
kSLCD\_RegulatedVolatgeTrim01,
   
kSLCD\_RegulatedVolatgeTrim02,
   
kSLCD\_RegulatedVolatgeTrim03,
   
kSLCD\_RegulatedVolatgeTrim04,
   
kSLCD\_RegulatedVolatgeTrim05,
   
kSLCD\_RegulatedVolatgeTrim06,
   
kSLCD\_RegulatedVolatgeTrim07,
   
kSLCD\_RegulatedVolatgeTrim08,
   
kSLCD\_RegulatedVolatgeTrim09,
   
kSLCD\_RegulatedVolatgeTrim10,
   
kSLCD\_RegulatedVolatgeTrim11,
   
kSLCD\_RegulatedVolatgeTrim12,
   
kSLCD\_RegulatedVolatgeTrim13,
   
kSLCD\_RegulatedVolatgeTrim14,
   
kSLCD\_RegulatedVolatgeTrim15 }  
*SLCD regulated voltage trim parameter, be used to meet the desired contrast.*
- enum `slcd_load_adjust_t` {
   
kSLCD\_LowLoadOrFastestClkSrc = 0U,
   
kSLCD\_LowLoadOrIntermediateClkSrc,
   
kSLCD\_HighLoadOrIntermediateClkSrc,
   
kSLCD\_HighLoadOrSlowestClkSrc }  
*SLCD load adjust to handle different LCD glass capacitance or configure the LCD charge pump clock source.*
- enum `slcd_clock_src_t` {
   
kSLCD\_DefaultClk = 0U,
   
kSLCD\_AlternateClk1 = 1U }  
*SLCD clock source.*
- enum `slcd_alt_clock_div_t` {
   
kSLCD\_AltClkDivFactor1 = 0U,
   
kSLCD\_AltClkDivFactor64,
   
kSLCD\_AltClkDivFactor256,
   
kSLCD\_AltClkDivFactor512 }  
*SLCD alternate clock divider.*
- enum `slcd_clock_prescaler_t` {

```
kSLCD_ClkPrescaler00 = 0U,
kSLCD_ClkPrescaler01,
kSLCD_ClkPrescaler02,
kSLCD_ClkPrescaler03,
kSLCD_ClkPrescaler04,
kSLCD_ClkPrescaler05,
kSLCD_ClkPrescaler06,
kSLCD_ClkPrescaler07 }
```

*SLCD clock prescaler to generate frame frequency.*

- enum `slcd_duty_cycle_t` {
 

```
kSLCD_1Div1DutyCycle = 0U,
kSLCD_1Div2DutyCycle,
kSLCD_1Div3DutyCycle,
kSLCD_1Div4DutyCycle,
kSLCD_1Div5DutyCycle,
kSLCD_1Div6DutyCycle,
kSLCD_1Div7DutyCycle,
kSLCD_1Div8DutyCycle }
```

*SLCD duty cycle.*

- enum `slcd_phase_type_t` {
 

```
kSLCD_NoPhaseActivate = 0x00U,
kSLCD_PhaseAActivate = 0x01U,
kSLCD_PhaseBActivate = 0x02U,
kSLCD_PhaseCActivate = 0x04U,
kSLCD_PhaseDActivate = 0x08U,
kSLCD_PhaseEActivate = 0x10U,
kSLCD_PhaseFActivate = 0x20U,
kSLCD_PhaseGActivate = 0x40U,
kSLCD_PhaseHActivate = 0x80U }
```

*SLCD segment phase type.*

- enum `slcd_phase_index_t` {
 

```
kSLCD_PhaseAIndex = 0x0U,
kSLCD_PhaseBIndex = 0x1U,
kSLCD_PhaseCIndex = 0x2U,
kSLCD_PhaseDIndex = 0x3U,
kSLCD_PhaseEIndex = 0x4U,
kSLCD_PhaseFIndex = 0x5U,
kSLCD_PhaseGIndex = 0x6U,
kSLCD_PhaseHIndex = 0x7U }
```

*SLCD segment phase bit index.*

- enum `slcd_display_mode_t` {
 

```
kSLCD_NormalMode = 0U,
kSLCD_AlternateMode,
kSLCD_BlinkMode }
```

*SLCD display mode.*

- enum `slcd_blink_mode_t` {

- ```
kSLCD_BlinkDisplayBlink = 0U,
kSLCD_AltDisplayBlink }
```

*SLCD blink mode.*
- enum `slcd_blink_rate_t` {  
`kSLCD_BlinkRate00` = 0U,  
`kSLCD_BlinkRate01`,  
`kSLCD_BlinkRate02`,  
`kSLCD_BlinkRate03`,  
`kSLCD_BlinkRate04`,  
`kSLCD_BlinkRate05`,  
`kSLCD_BlinkRate06`,  
`kSLCD_BlinkRate07` }

*SLCD blink rate.*
- enum `slcd_fault_detect_clock_prescaler_t` {  
`kSLCD_FaultSampleFreqDivider1` = 0U,  
`kSLCD_FaultSampleFreqDivider2`,  
`kSLCD_FaultSampleFreqDivider4`,  
`kSLCD_FaultSampleFreqDivider8`,  
`kSLCD_FaultSampleFreqDivider16`,  
`kSLCD_FaultSampleFreqDivider32`,  
`kSLCD_FaultSampleFreqDivider64`,  
`kSLCD_FaultSampleFreqDivider128` }

*SLCD fault detect clock prescaler.*
- enum `slcd_fault_detect_sample_window_width_t` {  
`kSLCD_FaultDetectWindowWidth4SampleClk` = 0U,  
`kSLCD_FaultDetectWindowWidth8SampleClk`,  
`kSLCD_FaultDetectWindowWidth16SampleClk`,  
`kSLCD_FaultDetectWindowWidth32SampleClk`,  
`kSLCD_FaultDetectWindowWidth64SampleClk`,  
`kSLCD_FaultDetectWindowWidth128SampleClk`,  
`kSLCD_FaultDetectWindowWidth256SampleClk`,  
`kSLCD_FaultDetectWindowWidth512SampleClk` }

*SLCD fault detect sample window width.*
- enum `slcd_interrupt_enable_t` { `kSLCD_FaultDetectCompleteInterrupt` = 1U }

*SLCD interrupt source.*
- enum `slcd_lowpower_behavior` {  
`kSLCD_EnabledInWaitStop` = 0,  
`kSLCD_EnabledInWaitOnly`,  
`kSLCD_EnabledInStopOnly`,  
`kSLCD_DisabledInWaitStop` }

*SLCD behavior in low power mode.*

## Driver version

- #define `FSL_SLCD_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 3))  
*SLCD driver version.*

## Initialization and deinitialization

- void **SLCD\_Init** (LCD\_Type \*base, **slcd\_config\_t** \*configure)  
*Initializes the SLCD, ungates the module clock, initializes the power setting, enables all used plane pins, and sets with interrupt and work mode with the configuration.*
- void **SLCD\_Deinit** (LCD\_Type \*base)  
*Deinitializes the SLCD module, gates the module clock, disables an interrupt, and displays the SLCD.*
- void **SLCD\_GetDefaultConfig** (**slcd\_config\_t** \*configure)  
*Gets the SLCD default configuration structure.*

## Plane Setting and Display Control

- static void **SLCD\_StartDisplay** (LCD\_Type \*base)  
*Enables the SLCD controller, starts generation, and displays the front plane and back plane waveform.*
- static void **SLCD\_StopDisplay** (LCD\_Type \*base)  
*Stops the SLCD controller.*
- void **SLCD\_StartBlinkMode** (LCD\_Type \*base, **slcd\_blink\_mode\_t** mode, **slcd\_blink\_rate\_t** rate)  
*Starts the SLCD blink mode.*
- static void **SLCD\_StopBlinkMode** (LCD\_Type \*base)  
*Stops the SLCD blink mode.*
- static void **SLCD\_SetBackPlanePhase** (LCD\_Type \*base, uint32\_t pinIdx, **slcd\_phase\_type\_t** phase)  
*Sets the SLCD back plane pin phase.*
- static void **SLCD\_SetFrontPlaneSegments** (LCD\_Type \*base, uint32\_t pinIdx, uint8\_t operation)  
*Sets the SLCD front plane segment operation for a front plane pin.*
- static void **SLCD\_SetFrontPlaneOnePhase** (LCD\_Type \*base, uint32\_t pinIdx, **slcd\_phase\_index\_t** phaseIdx, bool enable)  
*Sets one SLCD front plane pin for one phase.*
- static uint32\_t **SLCD\_GetFaultDetectCounter** (LCD\_Type \*base)  
*Gets the SLCD fault detect counter.*

## Interrupts.

- void **SLCD\_EnableInterrupts** (LCD\_Type \*base, uint32\_t mask)  
*Enables the SLCD interrupt.*
- void **SLCD\_DisableInterrupts** (LCD\_Type \*base, uint32\_t mask)  
*Disables the SLCD interrupt.*
- uint32\_t **SLCD\_GetInterruptStatus** (LCD\_Type \*base)  
*Gets the SLCD interrupt status flag.*
- void **SLCD\_ClearInterruptStatus** (LCD\_Type \*base, uint32\_t mask)  
*Clears the SLCD interrupt events status flag.*

## 29.4 Data Structure Documentation

### 29.4.1 struct slcd\_fault\_detect\_config\_t

#### Data Fields

- bool **faultDetectIntEnable**  
*Fault frame detection interrupt enable flag.*

- bool `faultDetectBackPlaneEnable`  
*True means the pin id fault detected is back plane otherwise front plane.*
- uint8\_t `faultDetectPinIndex`  
*Fault detected pin id from 0 to 63.*
- `slcd_fault_detect_clock_prescaler_t` `faultPrescaler`  
*Fault detect clock prescaler.*
- `slcd_fault_detect_sample_window_width_t` `width`  
*Fault detect sample window width.*

## Field Documentation

- (1) `bool slcd_fault_detect_config_t::faultDetectIntEnable`
- (2) `bool slcd_fault_detect_config_t::faultDetectBackPlaneEnable`
- (3) `uint8_t slcd_fault_detect_config_t::faultDetectPinIndex`
- (4) `slcd_fault_detect_clock_prescaler_t slcd_fault_detect_config_t::faultPrescaler`
- (5) `slcd_fault_detect_sample_window_width_t slcd_fault_detect_config_t::width`

## 29.4.2 struct `slcd_clock_config_t`

### Data Fields

- `slcd_clock_src_t clkSource`  
*Clock source.*
- `slcd_alt_clock_div_t altClkDivider`  
*The divider to divide the alternate clock used for alternate clock source.*
- `slcd_clock_prescaler_t clkPrescaler`  
*Clock prescaler.*

## Field Documentation

- (1) `slcd_clock_src_t slcd_clock_config_t::clkSource`  
"slcd\_clock\_src\_t" is recommended to be used. The SLCD is optimized to operate using a 32.768kHz clock input.
- (2) `slcd_alt_clock_div_t slcd_clock_config_t::altClkDivider`
- (3) `slcd_clock_prescaler_t slcd_clock_config_t::clkPrescaler`

## 29.4.3 struct `slcd_config_t`

### Data Fields

- `slcd_power_supply_option_t powerSupply`  
*Power supply option.*

- `slcd_regulated_voltage_trim_t voltageTrim`  
*Regulated voltage trim used for the internal regulator VIREG to adjust to facilitate contrast control.*
- `slcd_clock_config_t * clkConfig`  
*Clock configure.*
- `slcd_display_mode_t displayMode`  
*SLCD display mode.*
- `slcd_load_adjust_t loadAdjust`  
*Load adjust to handle glass capacitance.*
- `slcd_duty_cycle_t dutyCycle`  
*Duty cycle.*
- `slcd_lowpower_behavior lowPowerBehavior`  
*SLCD behavior in low power mode.*
- `uint32_t slcdLowPinEnabled`  
*Setting enabled SLCD pin 0 ~ pin 31.*
- `uint32_t slcdHighPinEnabled`  
*Setting enabled SLCD pin 32 ~ pin 63.*
- `uint32_t backPlaneLowPin`  
*Setting back plane pin 0 ~ pin 31.*
- `uint32_t backPlaneHighPin`  
*Setting back plane pin 32 ~ pin 63.*
- `slcd_fault_detect_config_t * faultConfig`  
*Fault frame detection configure.*

## Field Documentation

- (1) `slcd_power_supply_option_t slcd_config_t::powerSupply`
- (2) `slcd_regulated_voltage_trim_t slcd_config_t::voltageTrim`
- (3) `slcd_clock_config_t* slcd_config_t::clkConfig`
- (4) `slcd_display_mode_t slcd_config_t::displayMode`
- (5) `slcd_load_adjust_t slcd_config_t::loadAdjust`
- (6) `slcd_duty_cycle_t slcd_config_t::dutyCycle`
- (7) `slcd_lowpower_behavior slcd_config_t::lowPowerBehavior`
- (8) `uint32_t slcd_config_t::slcdLowPinEnabled`

Setting bit n to 1 means enable pin n.

- (9) `uint32_t slcd_config_t::slcdHighPinEnabled`

Setting bit n to 1 means enable pin (n + 32).

- (10) `uint32_t slcd_config_t::backPlaneLowPin`

Setting bit n to 1 means setting pin n as back plane. It should never have the same bit setting as the frontPlane Pin.

**(11) uint32\_t slcd\_config\_t::backPlaneHighPin**

Setting bit n to 1 means setting pin (n + 32) as back plane. It should never have the same bit setting as the frontPlane Pin.

**(12) slcd\_fault\_detect\_config\_t\* slcd\_config\_t::faultConfig**

If not requirement, set to NULL.

## 29.5 Macro Definition Documentation

### 29.5.1 #define FSL\_SLCD\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 3))

## 29.6 Enumeration Type Documentation

### 29.6.1 enum slcd\_power\_supply\_option\_t

Enumerator

*kSLCD\_InternalVll3UseChargePump* VLL3 connected to VDD internally, charge pump is used to generate VLL1 and VLL2.

*kSLCD\_ExternalVll3UseResistorBiasNetwork* VLL3 is driven externally and resistor bias network is used to generate VLL1 and VLL2.

*kSLCD\_ExteranlVll3UseChargePump* VLL3 is driven externally and charge pump is used to generate VLL1 and VLL2.

*kSLCD\_InternalVll1UseChargePump* VIREG is connected to VLL1 internally and charge pump is used to generate VLL2 and VLL3.

### 29.6.2 enum slcd\_regulated\_voltage\_trim\_t

Enumerator

*kSLCD\_RegulatedVolatgeTrim00* Increase the voltage to 0.91 V.

*kSLCD\_RegulatedVolatgeTrim01* Increase the voltage to 1.01 V.

*kSLCD\_RegulatedVolatgeTrim02* Increase the voltage to 0.96 V.

*kSLCD\_RegulatedVolatgeTrim03* Increase the voltage to 1.06 V.

*kSLCD\_RegulatedVolatgeTrim04* Increase the voltage to 0.93 V.

*kSLCD\_RegulatedVolatgeTrim05* Increase the voltage to 1.02 V.

*kSLCD\_RegulatedVolatgeTrim06* Increase the voltage to 0.98 V.

*kSLCD\_RegulatedVolatgeTrim07* Increase the voltage to 1.08 V.

*kSLCD\_RegulatedVolatgeTrim08* Increase the voltage to 0.92 V.

*kSLCD\_RegulatedVolatgeTrim09* Increase the voltage to 1.02 V.

*kSLCD\_RegulatedVolatgeTrim10* Increase the voltage to 0.97 V.

*kSLCD\_RegulatedVolatgeTrim11* Increase the voltage to 1.07 V.

*kSLCD\_RegulatedVolatgeTrim12* Increase the voltage to 0.94 V.

- kSLCD\_RegulatedVolatgeTrim13*** Increase the voltage to 1.05 V.
- kSLCD\_RegulatedVolatgeTrim14*** Increase the voltage to 0.99 V.
- kSLCD\_RegulatedVolatgeTrim15*** Increase the voltage to 1.09 V.

### 29.6.3 enum slcd\_load\_adjust\_t

Adjust the LCD glass capacitance if resistor bias network is enabled: kSLCD\_LowLoadOrFastestClkSrc - Low load (LCD glass capacitance 2000pF or lower. LCD or GPIO function can be used on VLL1,V-Ll2,Vcap1 and Vcap2 pins) kSLCD\_LowLoadOrIntermediateClkSrc - low load (LCD glass capacitance 2000pF or lower. LCD or GPIO function can be used on VLL1,VLL2,Vcap1 and Vcap2 pins) kSLCD\_HighLoadOrIntermediateClkSrc - high load (LCD glass capacitance 8000pF or lower. LCD or GPIO function can be used on Vcap1 and Vcap2 pins) kSLCD\_HighLoadOrSlowestClkSrc - high load (LCD glass capacitance 8000pF or lower LCD or GPIO function can be used on Vcap1 and Vcap2 pins) Adjust clock for charge pump if charge pump is enabled: kSLCD\_LowLoadOrFastestClkSrc - Fasten clock source (LCD glass capacitance 8000pF or 4000pF or lower if Fast Frame Rate is set) kSLCD\_LowLoadOrIntermediateClkSrc - Intermediate clock source (LCD glass capacitance 4000pF or 2000pF or lower if Fast Frame Rate is set) kSLCD\_HighLoadOrIntermediateClkSrc - Intermediate clock source (LCD glass capacitance 2000pF or 1000pF or lower if Fast Frame Rate is set) kSLCD\_HighLoadOrSlowestClkSrc - slowest clock source (LCD glass capacitance 1000pF or 500pF or lower if Fast Frame Rate is set)

Enumerator

- kSLCD\_LowLoadOrFastestClkSrc*** Adjust in low load or selects fastest clock.
- kSLCD\_LowLoadOrIntermediateClkSrc*** Adjust in low load or selects intermediate clock.
- kSLCD\_HighLoadOrIntermediateClkSrc*** Adjust in high load or selects intermediate clock.
- kSLCD\_HighLoadOrSlowestClkSrc*** Adjust in high load or selects slowest clock.

### 29.6.4 enum slcd\_clock\_src\_t

Enumerator

- kSLCD\_DefaultClk*** Select default clock ERCLK32K.
- kSLCD\_AlternateClk1*** Select alternate clock source 1 : MCGIRCLK.

### 29.6.5 enum slcd\_alt\_clock\_div\_t

Enumerator

- kSLCD\_AltClkDivFactor1*** No divide for alternate clock.
- kSLCD\_AltClkDivFactor64*** Divide alternate clock with factor 64.
- kSLCD\_AltClkDivFactor256*** Divide alternate clock with factor 256.
- kSLCD\_AltClkDivFactor512*** Divide alternate clock with factor 512.

**29.6.6 enum slcd\_clock\_prescaler\_t**

Enumerator

|                             |              |
|-----------------------------|--------------|
| <i>kSLCD_ClkPrescaler00</i> | Prescaler 0. |
| <i>kSLCD_ClkPrescaler01</i> | Prescaler 1. |
| <i>kSLCD_ClkPrescaler02</i> | Prescaler 2. |
| <i>kSLCD_ClkPrescaler03</i> | Prescaler 3. |
| <i>kSLCD_ClkPrescaler04</i> | Prescaler 4. |
| <i>kSLCD_ClkPrescaler05</i> | Prescaler 5. |
| <i>kSLCD_ClkPrescaler06</i> | Prescaler 6. |
| <i>kSLCD_ClkPrescaler07</i> | Prescaler 7. |

**29.6.7 enum slcd\_duty\_cycle\_t**

Enumerator

|                             |                              |
|-----------------------------|------------------------------|
| <i>kSLCD_IDiv1DutyCycle</i> | LCD use 1 BP 1/1 duty cycle. |
| <i>kSLCD_IDiv2DutyCycle</i> | LCD use 2 BP 1/2 duty cycle. |
| <i>kSLCD_IDiv3DutyCycle</i> | LCD use 3 BP 1/3 duty cycle. |
| <i>kSLCD_IDiv4DutyCycle</i> | LCD use 4 BP 1/4 duty cycle. |
| <i>kSLCD_IDiv5DutyCycle</i> | LCD use 5 BP 1/5 duty cycle. |
| <i>kSLCD_IDiv6DutyCycle</i> | LCD use 6 BP 1/6 duty cycle. |
| <i>kSLCD_IDiv7DutyCycle</i> | LCD use 7 BP 1/7 duty cycle. |
| <i>kSLCD_IDiv8DutyCycle</i> | LCD use 8 BP 1/8 duty cycle. |

**29.6.8 enum slcd\_phase\_type\_t**

Enumerator

|                              |                                  |
|------------------------------|----------------------------------|
| <i>kSLCD_NoPhaseActivate</i> | LCD waveform no phase activates. |
| <i>kSLCD_PhaseAActivate</i>  | LCD waveform phase A activates.  |
| <i>kSLCD_PhaseBActivate</i>  | LCD waveform phase B activates.  |
| <i>kSLCD_PhaseCActivate</i>  | LCD waveform phase C activates.  |
| <i>kSLCD_PhaseDActivate</i>  | LCD waveform phase D activates.  |
| <i>kSLCD_PhaseEActivate</i>  | LCD waveform phase E activates.  |
| <i>kSLCD_PhaseFActivate</i>  | LCD waveform phase F activates.  |
| <i>kSLCD_PhaseGActivate</i>  | LCD waveform phase G activates.  |
| <i>kSLCD_PhaseHActivate</i>  | LCD waveform phase H activates.  |

**29.6.9 enum slcd\_phase\_index\_t**

Enumerator

- kSLCD\_PhaseAIndex*** LCD phase A bit index.
- kSLCD\_PhaseBIndex*** LCD phase B bit index.
- kSLCD\_PhaseCIndex*** LCD phase C bit index.
- kSLCD\_PhaseDIndex*** LCD phase D bit index.
- kSLCD\_PhaseEIndex*** LCD phase E bit index.
- kSLCD\_PhaseFIndex*** LCD phase F bit index.
- kSLCD\_PhaseGIndex*** LCD phase G bit index.
- kSLCD\_PhaseHIndex*** LCD phase H bit index.

**29.6.10 enum slcd\_display\_mode\_t**

Enumerator

- kSLCD\_NormalMode*** LCD Normal display mode.
- kSLCD\_AlternateMode*** LCD Alternate display mode. For four back planes or less.
- kSLCD\_BankMode*** LCD Blank display mode.

**29.6.11 enum slcd\_blink\_mode\_t**

Enumerator

- kSLCD\_BankDisplayBlink*** Display blank during the blink period.
- kSLCD\_AltDisplayBlink*** Display alternate display during the blink period if duty cycle is lower than 5.

**29.6.12 enum slcd\_blink\_rate\_t**

Enumerator

- kSLCD\_BlinkRate00*** SLCD blink rate is LCD clock/(( $2^{12}$ )).
- kSLCD\_BlinkRate01*** SLCD blink rate is LCD clock/(( $2^{13}$ )).
- kSLCD\_BlinkRate02*** SLCD blink rate is LCD clock/(( $2^{14}$ )).
- kSLCD\_BlinkRate03*** SLCD blink rate is LCD clock/(( $2^{15}$ )).
- kSLCD\_BlinkRate04*** SLCD blink rate is LCD clock/(( $2^{16}$ )).
- kSLCD\_BlinkRate05*** SLCD blink rate is LCD clock/(( $2^{17}$ )).
- kSLCD\_BlinkRate06*** SLCD blink rate is LCD clock/(( $2^{18}$ )).
- kSLCD\_BlinkRate07*** SLCD blink rate is LCD clock/(( $2^{19}$ )).

**29.6.13 enum slcd\_fault\_detect\_clock\_prescaler\_t**

Enumerator

- kSLCD\_FaultSampleFreqDivider1* Fault detect sample clock frequency is 1/1 bus clock.
- kSLCD\_FaultSampleFreqDivider2* Fault detect sample clock frequency is 1/2 bus clock.
- kSLCD\_FaultSampleFreqDivider4* Fault detect sample clock frequency is 1/4 bus clock.
- kSLCD\_FaultSampleFreqDivider8* Fault detect sample clock frequency is 1/8 bus clock.
- kSLCD\_FaultSampleFreqDivider16* Fault detect sample clock frequency is 1/16 bus clock.
- kSLCD\_FaultSampleFreqDivider32* Fault detect sample clock frequency is 1/32 bus clock.
- kSLCD\_FaultSampleFreqDivider64* Fault detect sample clock frequency is 1/64 bus clock.
- kSLCD\_FaultSampleFreqDivider128* Fault detect sample clock frequency is 1/128 bus clock.

**29.6.14 enum slcd\_fault\_detect\_sample\_window\_width\_t**

Enumerator

- kSLCD\_FaultDetectWindowWidth4SampleClk* Sample window width is 4 sample clock cycles.
- kSLCD\_FaultDetectWindowWidth8SampleClk* Sample window width is 8 sample clock cycles.
- kSLCD\_FaultDetectWindowWidth16SampleClk* Sample window width is 16 sample clock cycles.
- kSLCD\_FaultDetectWindowWidth32SampleClk* Sample window width is 32 sample clock cycles.
- kSLCD\_FaultDetectWindowWidth64SampleClk* Sample window width is 64 sample clock cycles.
- kSLCD\_FaultDetectWindowWidth128SampleClk* Sample window width is 128 sample clock cycles.
- kSLCD\_FaultDetectWindowWidth256SampleClk* Sample window width is 256 sample clock cycles.
- kSLCD\_FaultDetectWindowWidth512SampleClk* Sample window width is 512 sample clock cycles.

**29.6.15 enum slcd\_interrupt\_enable\_t**

Enumerator

- kSLCD\_FaultDetectCompleteInterrupt* SLCD fault detection complete interrupt source.

**29.6.16 enum slcd\_lowpower\_behavior**

Enumerator

- kSLCD\_EnabledInWaitStop* SLCD works in wait and stop mode.
- kSLCD\_EnabledInWaitOnly* SLCD works in wait mode and is disabled in stop mode.
- kSLCD\_EnabledInStopOnly* SLCD works in stop mode and is disabled in wait mode.
- kSLCD\_DisabledInWaitStop* SLCD is disabled in stop mode and wait mode.

## 29.7 Function Documentation

### 29.7.1 void SLCD\_Init( LCD\_Type \* *base*, slcd\_config\_t \* *configure* )

Parameters

|                  |                                                                                                                                                                                                                                                                                                                                                              |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | SLCD peripheral base address.                                                                                                                                                                                                                                                                                                                                |
| <i>configure</i> | SLCD configuration pointer. For the configuration structure, many parameters have the default setting and the SLCD_Getdefaultconfig() is provided to get them. Use it verified for their applications. The others have no default settings, such as "clkConfig", and must be provided by the application before calling the <a href="#">SLCD_Init()</a> API. |

### 29.7.2 void SLCD\_Deinit( LCD\_Type \* *base* )

Parameters

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

### 29.7.3 void SLCD\_GetDefaultConfig( slcd\_config\_t \* *configure* )

The purpose of this API is to get default parameters of the configuration structure for the [SLCD\\_Init\(\)](#). Use these initialized parameters unchanged in [SLCD\\_Init\(\)](#) or modify fields of the structure before the calling [SLCD\\_Init\(\)](#). All default parameters of the configure structuration are listed.

```
config.displayMode      = kSLCD_NormalMode;
config.powerSupply     = kSLCD_InternalVll3UseChargePump;
config.voltageTrim    = kSLCD_RegulatedVolatgeTrim00;
config.lowPowerBehavior = kSLCD_EnabledInWaitStop;
config.interruptSrc   = 0;
config.faultConfig    = NULL;
config.frameFreqIntEnable = false;
```

Parameters

|                  |                                           |
|------------------|-------------------------------------------|
| <i>configure</i> | The SLCD configuration structure pointer. |
|------------------|-------------------------------------------|

### 29.7.4 static void SLCD\_StartDisplay( LCD\_Type \* *base* ) [inline], [static]

Parameters

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

### 29.7.5 static void SLCD\_StopDisplay( LCD\_Type \* *base* ) [inline], [static]

There is no waveform generator and all enabled pins only output a low value.

Parameters

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

### 29.7.6 void SLCD\_StartBlinkMode( LCD\_Type \* *base*, slcd\_blink\_mode\_t *mode*, slcd\_blink\_rate\_t *rate* )

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | SLCD peripheral base address. |
| <i>mode</i> | SLCD blink mode.              |
| <i>rate</i> | SLCD blink rate.              |

### 29.7.7 static void SLCD\_StopBlinkMode( LCD\_Type \* *base* ) [inline], [static]

Parameters

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

### 29.7.8 static void SLCD\_SetBackPlanePhase( LCD\_Type \* *base*, uint32\_t *pinIdx*, slcd\_phase\_type\_t *phase* ) [inline], [static]

This function sets the SLCD back plane pin phase. "kSLCD\_PhaseXActivate" setting means the phase X is active for the back plane pin. "kSLCD\_NoPhaseActivate" setting means there is no phase active for the back plane pin. For example, set the back plane pin 20 for phase A.

```
* SLCD_SetBackPlanePhase(LCD, 20, kSLCD_PhaseAActivate);
*
```

## Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | SLCD peripheral base address.                  |
| <i>pinIdx</i> | SLCD back plane pin index. Range from 0 to 63. |
| <i>phase</i>  | The phase activates for the back plane pin.    |

### 29.7.9 static void SLCD\_SetFrontPlaneSegments ( LCD\_Type \* *base*, uint32\_t *pinIdx*, uint8\_t *operation* ) [inline], [static]

This function sets the SLCD front plane segment on or off operation. Each bit turns on or off the segments associated with the front plane pin in the following pattern: HGFEDCBA (most significant bit controls segment H and least significant bit controls segment A). For example, turn on the front plane pin 20 for phase B and phase C.

```
* SLCD_SetFrontPlaneSegments(LCD, 20, (
    kSLCD_PhaseBActivate | kSLCD_PhaseCActivate));
*
```

## Parameters

|                  |                                                                                                                     |
|------------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | SLCD peripheral base address.                                                                                       |
| <i>pinIdx</i>    | SLCD back plane pin index. Range from 0 to 63.                                                                      |
| <i>operation</i> | The operation for the segment on the front plane pin. This is a logical OR of the enumeration :: slcd_phase_type_t. |

### 29.7.10 static void SLCD\_SetFrontPlaneOnePhase ( LCD\_Type \* *base*, uint32\_t *pinIdx*, slcd\_phase\_index\_t *phaseIdx*, bool *enable* ) [inline], [static]

This function can be used to set one phase on or off for the front plane pin. It can be call many times to set the plane pin for different phase indexes. For example, turn on the front plane pin 20 for phase B and phase C.

```
* SLCD_SetFrontPlaneOnePhase(LCD, 20,
    kSLCD_PhaseBIndex, true);
* SLCD_SetFrontPlaneOnePhase(LCD, 20,
    kSLCD_PhaseCIndex, true);
*
```

## Parameters

|                  |                                                                                                    |
|------------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>      | SLCD peripheral base address.                                                                      |
| <i>pinIndx</i>   | SLCD back plane pin index. Range from 0 to 63.                                                     |
| <i>phaseIndx</i> | The phase bit index <code>slcd_phase_index_t</code> .                                              |
| <i>enable</i>    | True to turn on the segment for phaseIndx phase false to turn off the segment for phaseIndx phase. |

**29.7.11 static uint32\_t SLCD\_GetFaultDetectCounter ( LCD\_Type \* *base* )  
[inline], [static]**

This function gets the number of samples inside the fault detection sample window.

## Parameters

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

## Returns

The fault detect counter. The maximum return value is 255. If the maximum 255 returns, the overflow may happen. Reconfigure the fault detect sample window and fault detect clock prescaler for proper sampling.

**29.7.12 void SLCD\_EnableInterrupts ( LCD\_Type \* *base*, uint32\_t *mask* )**

For example, to enable fault detect complete interrupt and frame frequency interrupt, for FSL\_FEATURE\_SLCD\_HAS\_FRAME\_FREQUENCY\_INTERRUPT enabled case, do the following.

```
*     SLCD_EnableInterrupts(LCD,
                           kSLCD_FaultDetectCompleteInterrupt | kSLCD_FrameFreqInterrupt);
*
```

## Parameters

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

|             |                                                                                                |
|-------------|------------------------------------------------------------------------------------------------|
| <i>mask</i> | SLCD interrupts to enable. This is a logical OR of the enumeration :: slcd_interrupt_enable_t. |
|-------------|------------------------------------------------------------------------------------------------|

### 29.7.13 void SLCD\_DisableInterrupts ( LCD\_Type \* *base*, uint32\_t *mask* )

For example, to disable fault detect complete interrupt and frame frequency interrupt, for FSL\_FEATURE\_SLCD\_HAS\_FRAME\_FREQUENCY\_INTERRUPT enabled case, do the following.

```
*     SLCD_DisableInterrupts(LCD,
*                           kSLCD_FaultDetectCompleteInterrupt | kSLCD_FrameFreqInterrupt);
*
```

Parameters

|             |                                                                                                 |
|-------------|-------------------------------------------------------------------------------------------------|
| <i>base</i> | SLCD peripheral base address.                                                                   |
| <i>mask</i> | SLCD interrupts to disable. This is a logical OR of the enumeration :: slcd_interrupt_enable_t. |

### 29.7.14 uint32\_t SLCD\_GetInterruptStatus ( LCD\_Type \* *base* )

Parameters

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

Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration :: slcd\_interrupt\_enable\_t.

### 29.7.15 void SLCD\_ClearInterruptStatus ( LCD\_Type \* *base*, uint32\_t *mask* )

Parameters

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

|             |                                                                                                                       |
|-------------|-----------------------------------------------------------------------------------------------------------------------|
| <i>mask</i> | SLCD interrupt source to be cleared. This is the logical OR of members of the enumeration :: slcd_interrupt_enable_t. |
|-------------|-----------------------------------------------------------------------------------------------------------------------|

# Chapter 30

## SMC: System Mode Controller Driver

### 30.1 Overview

The MCUXpresso SDK provides a peripheral driver for the System Mode Controller (SMC) module of MCUXpresso SDK devices. The SMC module sequences the system in and out of all low-power stop and run modes.

API functions are provided to configure the system for working in a dedicated power mode. For different power modes, `SMC_SetPowerModeXXX()` function accepts different parameters. System power mode state transitions are not available between power modes. For details about available transitions, see the power mode transitions section in the SoC reference manual.

### 30.2 Typical use case

#### 30.2.1 Enter wait or stop modes

SMC driver provides APIs to set MCU to different wait modes and stop modes. Pre and post functions are used for setting the modes. The pre functions and post functions are used as follows.

Disable/enable the interrupt through PRIMASK. This is an example use case. The application sets the wakeup interrupt and calls SMC function `SMC_SetPowerModeStop` to set the MCU to STOP mode, but the wakeup interrupt happens so quickly that the ISR completes before the function `SMC_SetPowerModeStop`. As a result, the MCU enters the STOP mode and never is woken up by the interrupt. In this use case, the application first disables the interrupt through PRIMASK, sets the wakeup interrupt, and enters the STOP mode. After wakeup, enable the interrupt through PRIMASK. The MCU can still be woken up by disabling the interrupt through PRIMASK. The pre and post functions handle the PRIMASK.

```
SMC_PreEnterStopModes();  
/* Enable the wakeup interrupt here. */  
SMC_SetPowerModeStop(SMC, kSMC_PartialStop);  
SMC_PostExitStopModes();
```

For legacy Kinetis, when entering stop modes, the flash speculation might be interrupted. As a result, the prefetched code or data might be broken. To make sure the flash is idle when entering the stop modes, smc driver allocates a RAM region, the code to enter stop modes are executed in RAM, thus the flash is idle and no prefetch is performed while entering stop modes. Application should make sure that, the rw data of `fsl_smci.c` is located in memory region which is not powered off in stop modes, especially LLS2 modes.

For STOP, VLPS, and LLS3, the whole RAM are powered up, so after woken up, the RAM function could continue executing. For VLLS mode, the system resets after woken up, the RAM content might be re-initialized. For LLS2 mode, only part of RAM are powered on, so application must make sure that, the

rw data of fsl\_smc.c is located in memory region which is not powered off, otherwise after woken up, the MCU could not get right code to execute.

## Data Structures

- struct `smc_power_mode_vlls_config_t`  
*SMC Very Low-Leakage Stop power mode configuration. [More...](#)*

## Enumerations

- enum `smc_power_mode_protection_t` {
   
`kSMC_AllowPowerModeVlls` = SMC\_PMPROT\_AVLLS\_MASK,  
`kSMC_AllowPowerModeVlp` = SMC\_PMPROT\_AVLP\_MASK,  
`kSMC_AllowPowerModeAll` }
   
*Power Modes Protection.*
- enum `smc_power_state_t` {
   
`kSMC_PowerStateRun` = 0x01U << 0U,  
`kSMC_PowerStateStop` = 0x01U << 1U,  
`kSMC_PowerStateVlpr` = 0x01U << 2U,  
`kSMC_PowerStateVlpw` = 0x01U << 3U,  
`kSMC_PowerStateVlps` = 0x01U << 4U,  
`kSMC_PowerStateVlls` = 0x01U << 6U }
   
*Power Modes in PMSTAT.*
- enum `smc_run_mode_t` {
   
`kSMC_RunNormal` = 0U,  
`kSMC_RunVlpr` = 2U }
   
*Run mode definition.*
- enum `smc_stop_mode_t` {
   
`kSMC_StopNormal` = 0U,  
`kSMC_StopVlps` = 2U,  
`kSMC_StopVlls` = 4U }
   
*Stop mode definition.*
- enum `smc_stop_submode_t` {
   
`kSMC_StopSub0` = 0U,  
`kSMC_StopSub1` = 1U,  
`kSMC_StopSub2` = 2U,  
`kSMC_StopSub3` = 3U }
   
*VLLS/LLS stop sub mode definition.*
- enum `smc_partial_stop_option_t` {
   
`kSMC_PartialStop` = 0U,  
`kSMC_PartialStop1` = 1U,  
`kSMC_PartialStop2` = 2U }
   
*Partial STOP option.*
- enum { `kStatus_SMC_StopAbort` = MAKE\_STATUS(kStatusGroup\_POWER, 0) }
   
*\_smc\_status, SMC configuration status.*

## Driver version

- #define **FSL\_SMC\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 7))  
*SMC driver version.*

## System mode controller APIs

- static void **SMC\_SetPowerModeProtection** (SMC\_Type \*base, uint8\_t allowedModes)  
*Configures all power mode protection settings.*
- static **smc\_power\_state\_t SMC\_GetPowerModeState** (SMC\_Type \*base)  
*Gets the current power mode status.*
- void **SMC\_PreEnterStopModes** (void)  
*Prepares to enter stop modes.*
- void **SMC\_PostExitStopModes** (void)  
*Recoveries after wake up from stop modes.*
- void **SMC\_PreEnterWaitModes** (void)  
*Prepares to enter wait modes.*
- void **SMC\_PostExitWaitModes** (void)  
*Recoveries after wake up from stop modes.*
- **status\_t SMC\_SetPowerModeRun** (SMC\_Type \*base)  
*Configures the system to RUN power mode.*
- **status\_t SMC\_SetPowerModeWait** (SMC\_Type \*base)  
*Configures the system to WAIT power mode.*
- **status\_t SMC\_SetPowerModeStop** (SMC\_Type \*base, **smc\_partial\_stop\_option\_t** option)  
*Configures the system to Stop power mode.*
- **status\_t SMC\_SetPowerModeVlpr** (SMC\_Type \*base)  
*Configures the system to VLPR power mode.*
- **status\_t SMC\_SetPowerModeVlpw** (SMC\_Type \*base)  
*Configures the system to VLPW power mode.*
- **status\_t SMC\_SetPowerModeVLps** (SMC\_Type \*base)  
*Configures the system to VLPS power mode.*
- **status\_t SMC\_SetPowerModeVlls** (SMC\_Type \*base, const **smc\_power\_mode\_vlls\_config\_t** \*config)  
*Configures the system to VLLS power mode.*

## 30.3 Data Structure Documentation

### 30.3.1 struct smc\_power\_mode\_vlls\_config\_t

#### Data Fields

- **smc\_stop\_submode\_t subMode**  
*Very Low-leakage Stop sub-mode.*
- bool **enablePorDetectInVlls0**  
*Enable Power on reset detect in VLLS mode.*

## 30.4 Enumeration Type Documentation

### 30.4.1 enum smc\_power\_mode\_protection\_t

Enumerator

- kSMC\_AllowPowerModeVlls* Allow Very-low-leakage Stop Mode.
- kSMC\_AllowPowerModeVlp* Allow Very-Low-power Mode.
- kSMC\_AllowPowerModeAll* Allow all power mode.

### 30.4.2 enum smc\_power\_state\_t

Enumerator

- kSMC\_PowerStateRun* 0000\_0001 - Current power mode is RUN
- kSMC\_PowerStateStop* 0000\_0010 - Current power mode is STOP
- kSMC\_PowerStateVlpr* 0000\_0100 - Current power mode is VLPR
- kSMC\_PowerStateVlpw* 0000\_1000 - Current power mode is VLPW
- kSMC\_PowerStateVlps* 0001\_0000 - Current power mode is VLPS
- kSMC\_PowerStateVlls* 0100\_0000 - Current power mode is VLLS

### 30.4.3 enum smc\_run\_mode\_t

Enumerator

- kSMC\_RunNormal* Normal RUN mode.
- kSMC\_RunVlpr* Very-low-power RUN mode.

### 30.4.4 enum smc\_stop\_mode\_t

Enumerator

- kSMC\_StopNormal* Normal STOP mode.
- kSMC\_StopVlps* Very-low-power STOP mode.
- kSMC\_StopVlls* Very-low-leakage Stop mode.

### 30.4.5 enum smc\_stop\_submode\_t

Enumerator

- kSMC\_StopSub0* Stop submode 0, for VLLS0/LLS0.
- kSMC\_StopSub1* Stop submode 1, for VLLS1/LLS1.
- kSMC\_StopSub2* Stop submode 2, for VLLS2/LLS2.
- kSMC\_StopSub3* Stop submode 3, for VLLS3/LLS3.

### 30.4.6 enum smc\_partial\_stop\_option\_t

Enumerator

*kSMC\_PartialStop* STOP - Normal Stop mode.

*kSMC\_PartialStop1* Partial Stop with both system and bus clocks disabled.

*kSMC\_PartialStop2* Partial Stop with system clock disabled and bus clock enabled.

### 30.4.7 anonymous enum

Enumerator

*kStatus\_SMC\_StopAbort* Entering Stop mode is abort.

## 30.5 Function Documentation

### 30.5.1 static void SMC\_SetPowerModeProtection ( **SMC\_Type** \* *base*, **uint8\_t** *allowedModes* ) [inline], [static]

This function configures the power mode protection settings for supported power modes in the specified chip family. The available power modes are defined in the **smc\_power\_mode\_protection\_t**. This should be done at an early system level initialization stage. See the reference manual for details. This register can only write once after the power reset.

The allowed modes are passed as bit map. For example, to allow LLS and VLLS, use **SMC\_SetPowerModeProtection(kSMC\_AllowPowerModeVlls | kSMC\_AllowPowerModeVlps)**. To allow all modes, use **SMC\_SetPowerModeProtection(kSMC\_AllowPowerModeAll)**.

Parameters

|                     |                                    |
|---------------------|------------------------------------|
| <i>base</i>         | SMC peripheral base address.       |
| <i>allowedModes</i> | Bitmap of the allowed power modes. |

### 30.5.2 static **smc\_power\_state\_t** SMC\_GetPowerModeState ( **SMC\_Type** \* *base* ) [inline], [static]

This function returns the current power mode status. After the application switches the power mode, it should always check the status to check whether it runs into the specified mode or not. The application should check this mode before switching to a different mode. The system requires that only certain modes can switch to other specific modes. See the reference manual for details and the **smc\_power\_state\_t** for information about the power status.

Parameters

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

Returns

Current power mode status.

### 30.5.3 void SMC\_PreEnterStopModes ( void )

This function should be called before entering STOP/VLPS/LLS/VLLS modes.

### 30.5.4 void SMC\_PostExitStopModes ( void )

This function should be called after wake up from STOP/VLPS/LLS/VLLS modes. It is used with [SMC\\_PreEnterStopModes](#).

### 30.5.5 void SMC\_PreEnterWaitModes ( void )

This function should be called before entering WAIT/VLPW modes.

### 30.5.6 void SMC\_PostExitWaitModes ( void )

This function should be called after wake up from WAIT/VLPW modes. It is used with [SMC\\_PreEnterWaitModes](#).

### 30.5.7 status\_t SMC\_SetPowerModeRun ( SMC\_Type \* *base* )

Parameters

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

Returns

SMC configuration error code.

### 30.5.8 status\_t SMC\_SetPowerModeWait ( SMC\_Type \* *base* )

Parameters

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

Returns

SMC configuration error code.

### 30.5.9 status\_t SMC\_SetPowerModeStop ( SMC\_Type \* *base*, smc\_partial\_stop\_option\_t *option* )

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SMC peripheral base address. |
| <i>option</i> | Partial Stop mode option.    |

Returns

SMC configuration error code.

### 30.5.10 status\_t SMC\_SetPowerModeVlpr ( SMC\_Type \* *base* )

Parameters

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

Returns

SMC configuration error code.

### 30.5.11 status\_t SMC\_SetPowerModeVlpw ( SMC\_Type \* *base* )

Parameters

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

Returns

SMC configuration error code.

### 30.5.12 status\_t SMC\_SetPowerModeVlps ( SMC\_Type \* *base* )

Parameters

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

Returns

SMC configuration error code.

### 30.5.13 status\_t SMC\_SetPowerModeVlls ( SMC\_Type \* *base*, const smc\_power\_mode\_vlls\_config\_t \* *config* )

Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>base</i>   | SMC peripheral base address.                 |
| <i>config</i> | The VLLS power mode configuration structure. |

Returns

SMC configuration error code.

# Chapter 31

## SPI: Serial Peripheral Interface Driver

### 31.1 Overview

#### Modules

- SPI CMSIS driver
- SPI DMA Driver
- SPI Driver
- SPI FreeRTOS driver

## 31.2 SPI Driver

### 31.2.1 Overview

SPI driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for SPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the SPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SPI functional operation groups provide the functional API set.

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the spi\_handle\_t as the first parameter. Initialize the handle by calling the [SPI\\_MasterTransferCreateHandle\(\)](#) or [SPI\\_SlaveTransferCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [SPI\\_MasterTransferNonBlocking\(\)](#) and [SPI\\_SlaveTransferNonBlocking\(\)](#) set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus\_SPI\_Idle status.

### 31.2.2 Typical use case

#### 31.2.2.1 SPI master transfer using an interrupt method

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

#### 31.2.2.2 SPI Send/receive using a DMA method

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

## Data Structures

- struct [spi\\_master\\_config\\_t](#)  
*SPI master user configure structure. [More...](#)*
- struct [spi\\_slave\\_config\\_t](#)  
*SPI slave user configure structure. [More...](#)*
- struct [spi\\_transfer\\_t](#)  
*SPI transfer structure. [More...](#)*
- struct [spi\\_master\\_handle\\_t](#)  
*SPI transfer handle structure. [More...](#)*

## Macros

- #define **SPI\_DUMMYDATA** (0xFFU)  
*SPI dummy transfer data, the data is sent while txBuff is NULL.*
- #define **SPI\_RETRY\_TIMES** 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/  
*Retry times for waiting flag.*

## TypeDefs

- typedef spi\_master\_handle\_t **spi\_slave\_handle\_t**  
*Slave handle is the same with master handle.*
- typedef void(\* **spi\_master\_callback\_t** )(SPI\_Type \*base, spi\_master\_handle\_t \*handle, **status\_t** status, void \*userData)  
*SPI master callback for finished transmit.*
- typedef void(\* **spi\_slave\_callback\_t** )(SPI\_Type \*base, **spi\_slave\_handle\_t** \*handle, **status\_t** status, void \*userData)  
*SPI master callback for finished transmit.*

## Enumerations

- enum {
 **kStatus\_SPI\_Busy** = MAKE\_STATUS(kStatusGroup\_SPI, 0),
 **kStatus\_SPI\_Idle** = MAKE\_STATUS(kStatusGroup\_SPI, 1),
 **kStatus\_SPI\_Error** = MAKE\_STATUS(kStatusGroup\_SPI, 2),
 **kStatus\_SPI\_Timeout** = MAKE\_STATUS(kStatusGroup\_SPI, 3) }
   
*Return status for the SPI driver.*
- enum **spi\_clock\_polarity\_t** {
 **kSPI\_ClockPolarityActiveHigh** = 0x0U,
 **kSPI\_ClockPolarityActiveLow** }
   
*SPI clock polarity configuration.*
- enum **spi\_clock\_phase\_t** {
 **kSPI\_ClockPhaseFirstEdge** = 0x0U,
 **kSPI\_ClockPhaseSecondEdge** }
   
*SPI clock phase configuration.*
- enum **spi\_shift\_direction\_t** {
 **kSPI\_MsbFirst** = 0x0U,
 **kSPI\_LsbFirst** }
   
*SPI data shifter direction options.*
- enum **spi\_ss\_output\_mode\_t** {
 **kSPI\_SlaveSelectAsGpio** = 0x0U,
 **kSPI\_SlaveSelectFaultInput** = 0x2U,
 **kSPI\_SlaveSelectAutomaticOutput** = 0x3U }
   
*SPI slave select output mode options.*
- enum **spi\_pin\_mode\_t** {

- ```
kSPI_PinModeNormal = 0x0U,
kSPI_PinModeInput = 0x1U,
kSPI_PinModeOutput = 0x3U }
```

*SPI pin mode options.*
- enum `spi_data_bitcount_mode_t` {
 

```
kSPI_8BitMode = 0x0U,
kSPI_16BitMode }
```

*SPI data length mode options.*
- enum `_spi_interrupt_enable` {
 

```
kSPI_RxFullAndModfInterruptEnable = 0x1U,
kSPI_TxEmptyInterruptEnable = 0x2U,
kSPI_MatchInterruptEnable = 0x4U,
kSPI_RxFifoNearFullInterruptEnable = 0x8U,
kSPI_TxFifoNearEmptyInterruptEnable = 0x10U }
```

*SPI interrupt sources.*
- enum `_spi_flags` {
 

```
kSPI_RxBufferFullFlag = SPI_S_SPRF_MASK,
kSPI_MatchFlag = SPI_S_SPMF_MASK,
kSPI_TxBufferEmptyFlag = SPI_S_SPTEF_MASK,
kSPI_ModeFaultFlag = SPI_S_MODF_MASK,
kSPI_RxFifoNearFullFlag = SPI_S_RNFULLF_MASK,
kSPI_TxFifoNearEmptyFlag = SPI_S_TNEAREF_MASK,
kSPI_TxFifoFullFlag = SPI_S_TXFULLF_MASK,
kSPI_RxFifoEmptyFlag = SPI_S_RFIFOEF_MASK,
kSPI_TxFifoError = SPI_CI_TXFERR_MASK << 8U,
kSPI_RxFifoError = SPI_CI_RXFERR_MASK << 8U,
kSPI_TxOverflow = SPI_CI_TXFOF_MASK << 8U,
kSPI_RxOverflow = SPI_CI_RXFOF_MASK << 8U }
```

*SPI status flags.*
- enum `spi_w1c_interrupt_t` {
 

```
kSPI_RxFifoFullClearInterrupt = SPI_CI_SPRFCI_MASK,
kSPI_TxFifoEmptyClearInterrupt = SPI_CI_SPTEFCI_MASK,
kSPI_RxNearFullClearInterrupt = SPI_CI_RNFULLFCI_MASK,
kSPI_TxNearEmptyClearInterrupt = SPI_CI_TNEAREFCI_MASK }
```

*SPI FIFO write-1-to-clear interrupt flags.*
- enum `spi_txfifo_watermark_t` {
 

```
kSPI_TxFifoOneFourthEmpty = 0,
kSPI_TxFifoOneHalfEmpty = 1 }
```

*SPI TX FIFO watermark settings.*
- enum `spi_rxfifo_watermark_t` {
 

```
kSPI_RxFifoThreeFourthsFull = 0,
kSPI_RxFifoOneHalfFull = 1 }
```

*SPI RX FIFO watermark settings.*
- enum `_spi_dma_enable_t` {
 

```
kSPI_TxDmaEnable = SPI_C2_TXDMAE_MASK,
kSPI_RxDmaEnable = SPI_C2_RXDMAE_MASK,
```

```
kSPI_DmaAllEnable = (SPI_C2_TXDMAE_MASK | SPI_C2_RXDMAE_MASK) }
```

*SPI DMA source.*

## Variables

- volatile uint8\_t **g\_spiDummyData** []
 

*Global variable for dummy data value setting.*

## Driver version

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

*SPI driver version.*

## Initialization and deinitialization

- void **SPI\_MasterGetDefaultConfig** (spi\_master\_config\_t \*config)
 

*Sets the SPI master configuration structure to default values.*
- void **SPI\_MasterInit** (SPI\_Type \*base, const spi\_master\_config\_t \*config, uint32\_t srcClock\_Hz)
 

*Initializes the SPI with master configuration.*
- void **SPI\_SlaveGetDefaultConfig** (spi\_slave\_config\_t \*config)
 

*Sets the SPI slave configuration structure to default values.*
- void **SPI\_SlaveInit** (SPI\_Type \*base, const spi\_slave\_config\_t \*config)
 

*Initializes the SPI with slave configuration.*
- void **SPI\_Deinit** (SPI\_Type \*base)
 

*De-initializes the SPI.*
- static void **SPI\_Enable** (SPI\_Type \*base, bool enable)
 

*Enables or disables the SPI.*

## Status

- uint32\_t **SPI\_GetStatusFlags** (SPI\_Type \*base)
 

*Gets the status flag.*
- static void **SPI\_ClearInterrupt** (SPI\_Type \*base, uint8\_t mask)
 

*Clear the interrupt if enable INCTRL.*

## Interrupts

- void **SPI\_EnableInterrupts** (SPI\_Type \*base, uint32\_t mask)
 

*Enables the interrupt for the SPI.*
- void **SPI\_DisableInterrupts** (SPI\_Type \*base, uint32\_t mask)
 

*Disables the interrupt for the SPI.*

## DMA Control

- static void **SPI\_EnableDMA** (SPI\_Type \*base, uint8\_t mask, bool enable)  
*Enables the DMA source for SPI.*
- static uint32\_t **SPI\_GetDataRegisterAddress** (SPI\_Type \*base)  
*Gets the SPI tx/rx data register address.*

## Bus Operations

- uint32\_t **SPIGetInstance** (SPI\_Type \*base)  
*Get the instance for SPI module.*
- static void **SPI\_SetPinMode** (SPI\_Type \*base, **spi\_pin\_mode\_t** pinMode)  
*Sets the pin mode for transfer.*
- void **SPI\_MasterSetBaudRate** (SPI\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the baud rate for SPI transfer.*
- static void **SPI\_SetMatchData** (SPI\_Type \*base, uint32\_t matchData)  
*Sets the match data for SPI.*
- void **SPI\_EnableFIFO** (SPI\_Type \*base, bool enable)  
*Enables or disables the FIFO if there is a FIFO.*
- **status\_t SPI\_WriteBlocking** (SPI\_Type \*base, uint8\_t \*buffer, size\_t size)  
*Sends a buffer of data bytes using a blocking method.*
- void **SPI\_WriteData** (SPI\_Type \*base, uint16\_t data)  
*Writes a data into the SPI data register.*
- uint16\_t **SPI\_ReadData** (SPI\_Type \*base)  
*Gets a data from the SPI data register.*
- void **SPI\_SetDummyData** (SPI\_Type \*base, uint8\_t dummyData)  
*Set up the dummy data.*

## Transactional

- void **SPI\_MasterTransferCreateHandle** (SPI\_Type \*base, **spi\_master\_handle\_t** \*handle, **spi\_master\_callback\_t** callback, void \*userData)  
*Initializes the SPI master handle.*
- **status\_t SPI\_MasterTransferBlocking** (SPI\_Type \*base, **spi\_transfer\_t** \*xfer)  
*Transfers a block of data using a polling method.*
- **status\_t SPI\_MasterTransferNonBlocking** (SPI\_Type \*base, **spi\_master\_handle\_t** \*handle, **spi\_transfer\_t** \*xfer)  
*Performs a non-blocking SPI interrupt transfer.*
- **status\_t SPI\_MasterTransferGetCount** (SPI\_Type \*base, **spi\_master\_handle\_t** \*handle, size\_t \*count)  
*Gets the bytes of the SPI interrupt transferred.*
- void **SPI\_MasterTransferAbort** (SPI\_Type \*base, **spi\_master\_handle\_t** \*handle)  
*Aborts an SPI transfer using interrupt.*
- void **SPI\_MasterTransferHandleIRQ** (SPI\_Type \*base, **spi\_master\_handle\_t** \*handle)  
*Interrupts the handler for the SPI.*
- void **SPI\_SlaveTransferCreateHandle** (SPI\_Type \*base, **spi\_slave\_handle\_t** \*handle, **spi\_slave\_callback\_t** callback, void \*userData)  
*Initializes the SPI slave handle.*

- `status_t SPI_SlaveTransferNonBlocking (SPI_Type *base, spi_slave_handle_t *handle, spi_transfer_t *xfer)`  
*Performs a non-blocking SPI slave interrupt transfer.*
- `static status_t SPI_SlaveTransferGetCount (SPI_Type *base, spi_slave_handle_t *handle, size_t *count)`  
*Gets the bytes of the SPI interrupt transferred.*
- `static void SPI_SlaveTransferAbort (SPI_Type *base, spi_slave_handle_t *handle)`  
*Aborts an SPI slave transfer using interrupt.*
- `void SPI_SlaveTransferHandleIRQ (SPI_Type *base, spi_slave_handle_t *handle)`  
*Interrupts a handler for the SPI slave.*

### 31.2.3 Data Structure Documentation

#### 31.2.3.1 struct spi\_master\_config\_t

##### Data Fields

- `bool enableMaster`  
*Enable SPI at initialization time.*
- `bool enableStopInWaitMode`  
*SPI stop in wait mode.*
- `spi_clock_polarity_t polarity`  
*Clock polarity.*
- `spi_clock_phase_t phase`  
*Clock phase.*
- `spi_shift_direction_t direction`  
*MSB or LSB.*
- `spi_data_bitcount_mode_t dataMode`  
*8bit or 16bit mode*
- `spi_tx_fifo_watermark_t txWatermark`  
*Tx watermark settings.*
- `spi_rx_fifo_watermark_t rxWatermark`  
*Rx watermark settings.*
- `spi_ss_output_mode_t outputMode`  
*SS pin setting.*
- `spi_pin_mode_t pinMode`  
*SPI pin mode select.*
- `uint32_t baudRate_Bps`  
*Baud Rate for SPI in Hz.*

#### 31.2.3.2 struct spi\_slave\_config\_t

##### Data Fields

- `bool enableSlave`  
*Enable SPI at initialization time.*
- `bool enableStopInWaitMode`  
*SPI stop in wait mode.*

- `spi_clock_polarity_t` polarity  
*Clock polarity.*
- `spi_clock_phase_t` phase  
*Clock phase.*
- `spi_shift_direction_t` direction  
*MSB or LSB.*
- `spi_data_bitcount_mode_t` dataMode  
*8bit or 16bit mode*
- `spi_txfifo_watermark_t` txWatermark  
*Tx watermark settings.*
- `spi_rx_fifo_watermark_t` rxWatermark  
*Rx watermark settings.*
- `spi_pin_mode_t` pinMode  
*SPI pin mode select.*

### 31.2.3.3 struct spi\_transfer\_t

#### Data Fields

- `uint8_t * txData`  
*Send buffer.*
- `uint8_t * rxData`  
*Receive buffer.*
- `size_t dataSize`  
*Transfer bytes.*
- `uint32_t flags`  
*SPI control flag, useless to SPI.*

#### Field Documentation

(1) `uint32_t spi_transfer_t::flags`

### 31.2.3.4 struct \_spi\_master\_handle

#### Data Fields

- `uint8_t *volatile txData`  
*Transfer buffer.*
- `uint8_t *volatile rxData`  
*Receive buffer.*
- `volatile size_t txRemainingBytes`  
*Send data remaining in bytes.*
- `volatile size_t rxRemainingBytes`  
*Receive data remaining in bytes.*
- `volatile uint32_t state`  
*SPI internal state.*
- `size_t transferSize`  
*Bytes to be transferred.*
- `uint8_t bytePerFrame`  
*SPI mode, 2bytes or 1byte in a frame.*

- `uint8_t watermark`  
*Watermark value for SPI transfer.*
- `spi_master_callback_t callback`  
*SPI callback.*
- `void * userData`  
*Callback parameter.*

### 31.2.4 Macro Definition Documentation

**31.2.4.1 #define FSL\_SPI\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))**

**31.2.4.2 #define SPI\_DUMMYDATA (0xFFU)**

**31.2.4.3 #define SPI\_RETRY\_TIMES 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/**

### 31.2.5 Enumeration Type Documentation

#### 31.2.5.1 anonymous enum

Enumerator

*kStatus\_SPI\_Busy* SPI bus is busy.

*kStatus\_SPI\_Idle* SPI is idle.

*kStatus\_SPI\_Error* SPI error.

*kStatus\_SPI\_Timeout* SPI timeout polling status flags.

#### 31.2.5.2 enum spi\_clock\_polarity\_t

Enumerator

*kSPI\_ClockPolarityActiveHigh* Active-high SPI clock (idles low).

*kSPI\_ClockPolarityActiveLow* Active-low SPI clock (idles high).

#### 31.2.5.3 enum spi\_clock\_phase\_t

Enumerator

*kSPI\_ClockPhaseFirstEdge* First edge on SPSCK occurs at the middle of the first cycle of a data transfer.

*kSPI\_ClockPhaseSecondEdge* First edge on SPSCK occurs at the start of the first cycle of a data transfer.

### 31.2.5.4 enum spi\_shift\_direction\_t

Enumerator

***kSPI\_MsbFirst*** Data transfers start with most significant bit.

***kSPI\_LsbFirst*** Data transfers start with least significant bit.

### 31.2.5.5 enum spi\_ss\_output\_mode\_t

Enumerator

***kSPI\_SlaveSelectAsGpio*** Slave select pin configured as GPIO.

***kSPI\_SlaveSelectFaultInput*** Slave select pin configured for fault detection.

***kSPI\_SlaveSelectAutomaticOutput*** Slave select pin configured for automatic SPI output.

### 31.2.5.6 enum spi\_pin\_mode\_t

Enumerator

***kSPI\_PinModeNormal*** Pins operate in normal, single-direction mode.

***kSPI\_PinModeInput*** Bidirectional mode. Master: MOSI pin is input; Slave: MISO pin is input.

***kSPI\_PinModeOutput*** Bidirectional mode. Master: MOSI pin is output; Slave: MISO pin is output.

### 31.2.5.7 enum spi\_data\_bitcount\_mode\_t

Enumerator

***kSPI\_8BitMode*** 8-bit data transmission mode

***kSPI\_16BitMode*** 16-bit data transmission mode

### 31.2.5.8 enum \_spi\_interrupt\_enable

Enumerator

***kSPI\_RxFullAndModfInterruptEnable*** Receive buffer full (SPRF) and mode fault (MODF) interrupt.

***kSPI\_TxEmptyInterruptEnable*** Transmit buffer empty interrupt.

***kSPI\_MatchInterruptEnable*** Match interrupt.

***kSPI\_RxFifoNearFullInterruptEnable*** Receive FIFO nearly full interrupt.

***kSPI\_TxFifoNearEmptyInterruptEnable*** Transmit FIFO nearly empty interrupt.

### 31.2.5.9 enum \_spi\_flags

Enumerator

- kSPI\_RxBufferFullFlag* Read buffer full flag.
- kSPI\_MatchFlag* Match flag.
- kSPI\_TxBufferEmptyFlag* Transmit buffer empty flag.
- kSPI\_ModeFaultFlag* Mode fault flag.
- kSPI\_RxFifoNearFullFlag* Rx FIFO near full.
- kSPI\_TxFifoNearEmptyFlag* Tx FIFO near empty.
- kSPI\_TxFifoFullFlag* Tx FIFO full.
- kSPI\_RxFifoEmptyFlag* Rx FIFO empty.
- kSPI\_TxFifoError* Tx FIFO error.
- kSPI\_RxFifoError* Rx FIFO error.
- kSPI\_TxOverflow* Tx FIFO Overflow.
- kSPI\_RxOverflow* Rx FIFO Overflow.

### 31.2.5.10 enum spi\_w1c\_interrupt\_t

Enumerator

- kSPI\_RxFifoFullClearInterrupt* Receive FIFO full interrupt.
- kSPI\_TxFifoEmptyClearInterrupt* Transmit FIFO empty interrupt.
- kSPI\_RxNearFullClearInterrupt* Receive FIFO nearly full interrupt.
- kSPI\_TxNearEmptyClearInterrupt* Transmit FIFO nearly empty interrupt.

### 31.2.5.11 enum spi\_txfifo\_watermark\_t

Enumerator

- kSPI\_TxFifoOneFourthEmpty* SPI tx watermark at 1/4 FIFO size.
- kSPI\_TxFifoOneHalfEmpty* SPI tx watermark at 1/2 FIFO size.

### 31.2.5.12 enum spi\_rxfifo\_watermark\_t

Enumerator

- kSPI\_RxFifoThreeFourthsFull* SPI rx watermark at 3/4 FIFO size.
- kSPI\_RxFifoOneHalfFull* SPI rx watermark at 1/2 FIFO size.

### 31.2.5.13 enum \_spi\_dma\_enable\_t

Enumerator

- kSPI\_TxDmaEnable* Tx DMA request source.
- kSPI\_RxDmaEnable* Rx DMA request source.
- kSPI\_DmaAllEnable* All DMA request source.

## 31.2.6 Function Documentation

### 31.2.6.1 void SPI\_MasterGetDefaultConfig ( spi\_master\_config\_t \* config )

The purpose of this API is to get the configuration structure initialized for use in [SPI\\_MasterInit\(\)](#). User may use the initialized structure unchanged in [SPI\\_MasterInit\(\)](#), or modify some fields of the structure before calling [SPI\\_MasterInit\(\)](#). After calling this API, the master is ready to transfer. Example:

```
spi_master_config_t config;
SPI_MasterGetDefaultConfig(&config);
```

Parameters

<i>config</i>	pointer to master config structure
---------------	------------------------------------

### 31.2.6.2 void SPI\_MasterInit ( SPI\_Type \* base, const spi\_master\_config\_t \* config, uint32\_t srcClock\_Hz )

The configuration structure can be filled by user from scratch, or be set with default values by [SPI\\_MasterGetDefaultConfig\(\)](#). After calling this API, the slave is ready to transfer. Example

```
spi_master_config_t config = {
    .baudRate_Bps = 400000,
    ...
};
SPI_MasterInit(SPI0, &config);
```

Parameters

<i>base</i>	SPI base pointer
-------------	------------------

<i>config</i>	pointer to master configuration structure
<i>srcClock_Hz</i>	Source clock frequency.

### 31.2.6.3 void SPI\_SlaveGetDefaultConfig ( spi\_slave\_config\_t \* *config* )

The purpose of this API is to get the configuration structure initialized for use in [SPI\\_SlaveInit\(\)](#). Modify some fields of the structure before calling [SPI\\_SlaveInit\(\)](#). Example:

```
spi_slave_config_t config;
SPI_SlaveGetDefaultConfig(&config);
```

Parameters

<i>config</i>	pointer to slave configuration structure
---------------	--

### 31.2.6.4 void SPI\_SlaveInit ( SPI\_Type \* *base*, const spi\_slave\_config\_t \* *config* )

The configuration structure can be filled by user from scratch or be set with default values by [SPI\\_SlaveGetDefaultConfig\(\)](#). After calling this API, the slave is ready to transfer. Example

```
spi_slave_config_t config = {
.polarity = kSPIClockPolarity_ActiveHigh;
.phase = kSPIClockPhase_FirstEdge;
.direction = kSPIMsbFirst;
...
};
SPI_MasterInit(SPI0, &config);
```

Parameters

<i>base</i>	SPI base pointer
<i>config</i>	pointer to master configuration structure

### 31.2.6.5 void SPI\_Deinit ( SPI\_Type \* *base* )

Calling this API resets the SPI module, gates the SPI clock. The SPI module can't work unless calling the SPI\_MasterInit/SPI\_SlaveInit to initialize module.

Parameters

<i>base</i>	SPI base pointer
-------------	------------------

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

Parameters

<i>base</i>	SPI base pointer
<i>enable</i>	pass true to enable module, false to disable module

### 31.2.6.7 uint32\_t SPI\_GetStatusFlags ( SPI\_Type \* *base* )

Parameters

<i>base</i>	SPI base pointer
-------------	------------------

Returns

SPI Status, use status flag to AND [\\_spi\\_flags](#) could get the related status.

### 31.2.6.8 static void SPI\_ClearInterrupt ( SPI\_Type \* *base*, uint8\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	SPI base pointer
<i>mask</i>	Interrupt need to be cleared The parameter could be any combination of the following values: <ul style="list-style-type: none"><li>• kSPI_RxFullAndModfInterruptEnable</li><li>• kSPI_TxEmptyInterruptEnable</li><li>• kSPI_MatchInterruptEnable</li><li>• kSPI_RxFifoNearFullInterruptEnable</li><li>• kSPI_TxFifoNearEmptyInterruptEnable</li></ul>

### 31.2.6.9 void SPI\_EnableInterrupts ( SPI\_Type \* *base*, uint32\_t *mask* )

Parameters

<i>base</i>	SPI base pointer
<i>mask</i>	SPI interrupt source. The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• kSPI_RxFullAndModfInterruptEnable</li> <li>• kSPI_TxEmptyInterruptEnable</li> <li>• kSPI_MatchInterruptEnable</li> <li>• kSPI_RxFifoNearFullInterruptEnable</li> <li>• kSPI_TxFifoNearEmptyInterruptEnable</li> </ul>

### 31.2.6.10 void SPI\_DisableInterrupts ( SPI\_Type \* *base*, uint32\_t *mask* )

Parameters

<i>base</i>	SPI base pointer
<i>mask</i>	SPI interrupt source. The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• kSPI_RxFullAndModfInterruptEnable</li> <li>• kSPI_TxEmptyInterruptEnable</li> <li>• kSPI_MatchInterruptEnable</li> <li>• kSPI_RxFifoNearFullInterruptEnable</li> <li>• kSPI_TxFifoNearEmptyInterruptEnable</li> </ul>

### 31.2.6.11 static void SPI\_EnableDMA ( SPI\_Type \* *base*, uint8\_t *mask*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	SPI base pointer
<i>mask</i>	SPI DMA source.
<i>enable</i>	True means enable DMA, false means disable DMA

### 31.2.6.12 static uint32\_t SPI\_GetDataRegisterAddress ( SPI\_Type \* *base* ) [inline], [static]

This API is used to provide a transfer address for the SPI DMA transfer configuration.

Parameters

<i>base</i>	SPI base pointer
-------------	------------------

Returns

data register address

### 31.2.6.13 `uint32_t SPI_GetInstance ( SPI_Type * base )`

Parameters

<i>base</i>	SPI base address
-------------	------------------

### 31.2.6.14 `static void SPI_SetPinMode ( SPI_Type * base, spi_pin_mode_t pinMode ) [inline], [static]`

Parameters

<i>base</i>	SPI base pointer
<i>pinMode</i>	pin mode for transfer AND _spi_pin_mode could get the related configuration.

### 31.2.6.15 `void SPI_MasterSetBaudRate ( SPI_Type * base, uint32_t baudRate_Bps, uint32_t srcClock_Hz )`

This is only used in master.

Parameters

<i>base</i>	SPI base pointer
<i>baudRate_Bps</i>	baud rate needed in Hz.
<i>srcClock_Hz</i>	SPI source clock frequency in Hz.

### 31.2.6.16 `static void SPI_SetMatchData ( SPI_Type * base, uint32_t matchData ) [inline], [static]`

The match data is a hardware comparison value. When the value received in the SPI receive data buffer equals the hardware comparison value, the SPI Match Flag in the S register (S[SPMF]) sets. This can also generate an interrupt if the enable bit sets.

Parameters

<i>base</i>	SPI base pointer
<i>matchData</i>	Match data.

### 31.2.6.17 void SPI\_EnableFIFO ( SPI\_Type \* *base*, bool *enable* )

Parameters

<i>base</i>	SPI base pointer
<i>enable</i>	True means enable FIFO, false means disable FIFO.

### 31.2.6.18 status\_t SPI\_WriteBlocking ( SPI\_Type \* *base*, uint8\_t \* *buffer*, size\_t *size* )

Note

This function blocks via polling until all bytes have been sent.

Parameters

<i>base</i>	SPI base pointer
<i>buffer</i>	The data bytes to send
<i>size</i>	The number of data bytes to send

Returns

kStatus\_SPI\_Timeout The transfer timed out and was aborted.

### 31.2.6.19 void SPI\_WriteData ( SPI\_Type \* *base*, uint16\_t *data* )

Parameters

<i>base</i>	SPI base pointer
<i>data</i>	needs to be write.

### 31.2.6.20 uint16\_t SPI\_ReadData ( SPI\_Type \* *base* )

Parameters

<i>base</i>	SPI base pointer
-------------	------------------

Returns

Data in the register.

### 31.2.6.21 void SPI\_SetDummyData ( SPI\_Type \* *base*, uint8\_t *dummyData* )

Parameters

<i>base</i>	SPI peripheral address.
<i>dummyData</i>	Data to be transferred when tx buffer is NULL.

### 31.2.6.22 void SPI\_MasterTransferCreateHandle ( SPI\_Type \* *base*, spi\_master\_handle\_t \* *handle*, spi\_master\_callback\_t *callback*, void \* *userData* )

This function initializes the SPI master handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, call this API once to get the initialized handle.

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	SPI handle pointer.
<i>callback</i>	Callback function.
<i>userData</i>	User data.

### 31.2.6.23 status\_t SPI\_MasterTransferBlocking ( SPI\_Type \* *base*, spi\_transfer\_t \* *xfer* )

Parameters

<i>base</i>	SPI base pointer
<i>xfer</i>	pointer to spi_xfer_config_t structure

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.

### 31.2.6.24 status\_t SPI\_MasterTransferNonBlocking ( **SPI\_Type** \* *base*, **spi\_master\_handle\_t** \* *handle*, **spi\_transfer\_t** \* *xfer* )

Note

The API immediately returns after transfer initialization is finished. Call SPI\_GetStatusIRQ() to get the transfer status.

If SPI transfer data frame size is 16 bits, the transfer size cannot be an odd number.

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	pointer to <b>spi_master_handle_t</b> structure which stores the transfer state
<i>xfer</i>	pointer to <b>spi_xfer_config_t</b> structure

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.
<i>kStatus_SPI_Busy</i>	SPI is not idle, is running another transfer.

### 31.2.6.25 status\_t SPI\_MasterTransferGetCount ( **SPI\_Type** \* *base*, **spi\_master\_handle\_t** \* *handle*, **size\_t** \* *count* )

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	Pointer to SPI transfer handle, this should be a static variable.
<i>count</i>	Transferred bytes of SPI master.

Return values

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

### 31.2.6.26 void SPI\_MasterTransferAbort ( *SPI\_Type* \* *base*, *spi\_master\_handle\_t* \* *handle* )

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	Pointer to SPI transfer handle, this should be a static variable.

### 31.2.6.27 void SPI\_MasterTransferHandleIRQ ( *SPI\_Type* \* *base*, *spi\_master\_handle\_t* \* *handle* )

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	pointer to <i>spi_master_handle_t</i> structure which stores the transfer state.

### 31.2.6.28 void SPI\_SlaveTransferCreateHandle ( *SPI\_Type* \* *base*, *spi\_slave\_handle\_t* \* *handle*, *spi\_slave\_callback\_t* *callback*, *void* \* *userData* )

This function initializes the SPI slave handle which can be used for other SPI slave transactional APIs. Usually, for a specified SPI instance, call this API once to get the initialized handle.

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	SPI handle pointer.
<i>callback</i>	Callback function.

<i>userData</i>	User data.
-----------------	------------

### 31.2.6.29 **status\_t SPI\_SlaveTransferNonBlocking ( SPI\_Type \* *base*, spi\_slave\_handle\_t \* *handle*, spi\_transfer\_t \* *xfer* )**

Note

The API returns immediately after the transfer initialization is finished. Call SPI\_GetStatusIRQ() to get the transfer status.

If SPI transfer data frame size is 16 bits, the transfer size cannot be an odd number.

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	pointer to spi_slave_handle_t structure which stores the transfer state
<i>xfer</i>	pointer to spi_xfer_config_t structure

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.
<i>kStatus_SPI_Busy</i>	SPI is not idle, is running another transfer.

### 31.2.6.30 **static status\_t SPI\_SlaveTransferGetCount ( SPI\_Type \* *base*, spi\_slave\_handle\_t \* *handle*, size\_t \* *count* ) [inline], [static]**

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	Pointer to SPI transfer handle, this should be a static variable.
<i>count</i>	Transferred bytes of SPI slave.

Return values

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

31.2.6.31 **static void SPI\_SlaveTransferAbort ( SPI\_Type \* *base*, spi\_slave\_handle\_t \* *handle* ) [inline], [static]**

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	Pointer to SPI transfer handle, this should be a static variable.

### 31.2.6.32 void SPI\_SlaveTransferHandleIRQ ( SPI\_Type \* *base*, spi\_slave\_handle\_t \* *handle* )

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	pointer to spi_slave_handle_t structure which stores the transfer state

## 31.2.7 Variable Documentation

### 31.2.7.1 volatile uint8\_t g\_spiDummyData[]

## 31.3 SPI DMA Driver

### 31.3.1 Overview

This section describes the programming interface of the SPI DMA driver.

## Data Structures

- struct `spi_dma_handle_t`

*SPI DMA transfer handle, users should not touch the content of the handle. [More...](#)*

## TypeDefs

- typedef void(\* `spi_dma_callback_t` )(SPI\_Type \*base, spi\_dma\_handle\_t \*handle, `status_t` status, void \*userData)

*SPI DMA callback called at the end of transfer.*

## Driver version

- #define `FSL_SPI_DMA_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 1)`)

*SPI DMA driver version.*

## DMA Transactional

- void `SPI_MasterTransferCreateHandleDMA` (SPI\_Type \*base, spi\_dma\_handle\_t \*handle, `spi_dma_callback_t` callback, void \*userData, `dma_handle_t` \*txHandle, `dma_handle_t` \*rxHandle)  
*Initialize the SPI master DMA handle.*
- `status_t SPI_MasterTransferDMA` (SPI\_Type \*base, spi\_dma\_handle\_t \*handle, `spi_transfer_t` \*xfer)  
*Perform a non-blocking SPI transfer using DMA.*
- void `SPI_MasterTransferAbortDMA` (SPI\_Type \*base, spi\_dma\_handle\_t \*handle)  
*Abort a SPI transfer using DMA.*
- `status_t SPI_MasterTransferGetCountDMA` (SPI\_Type \*base, spi\_dma\_handle\_t \*handle, size\_t \*count)  
*Get the transferred bytes for SPI slave DMA.*
- static void `SPI_SlaveTransferCreateHandleDMA` (SPI\_Type \*base, spi\_dma\_handle\_t \*handle, `spi_dma_callback_t` callback, void \*userData, `dma_handle_t` \*txHandle, `dma_handle_t` \*rxHandle)  
*Initialize the SPI slave DMA handle.*
- static `status_t SPI_SlaveTransferDMA` (SPI\_Type \*base, spi\_dma\_handle\_t \*handle, `spi_transfer_t` \*xfer)  
*Perform a non-blocking SPI transfer using DMA.*
- static void `SPI_SlaveTransferAbortDMA` (SPI\_Type \*base, spi\_dma\_handle\_t \*handle)  
*Abort a SPI transfer using DMA.*
- static `status_t SPI_SlaveTransferGetCountDMA` (SPI\_Type \*base, spi\_dma\_handle\_t \*handle, size\_t \*count)

*Get the transferred bytes for SPI slave DMA.*

### 31.3.2 Data Structure Documentation

#### 31.3.2.1 struct \_spi\_dma\_handle

##### Data Fields

- bool **txInProgress**  
*Send transfer finished.*
- bool **rxInProgress**  
*Receive transfer finished.*
- **dma\_handle\_t \* txHandle**  
*DMA handler for SPI send.*
- **dma\_handle\_t \* rxHandle**  
*DMA handler for SPI receive.*
- uint8\_t **bytesPerFrame**  
*Bytes in a frame for SPI transfer.*
- **spi\_dma\_callback\_t callback**  
*Callback for SPI DMA transfer.*
- void \* **userData**  
*User Data for SPI DMA callback.*
- uint32\_t **state**  
*Internal state of SPI DMA transfer.*
- size\_t **transferSize**  
*Bytes need to be transfer.*

### 31.3.3 Macro Definition Documentation

#### 31.3.3.1 #define FSL\_SPI\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))

### 31.3.4 Typedef Documentation

#### 31.3.4.1 **typedef void(\* spi\_dma\_callback\_t)(SPI\_Type \*base, spi\_dma\_handle\_t \*handle, status\_t status, void \*userData)**

### 31.3.5 Function Documentation

#### 31.3.5.1 **void SPI\_MasterTransferCreateHandleDMA ( SPI\_Type \* base, spi\_dma\_handle\_t \* handle, spi\_dma\_callback\_t callback, void \* userData, dma\_handle\_t \* txHandle, dma\_handle\_t \* rxHandle )**

This function initializes the SPI master DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	SPI handle pointer.
<i>callback</i>	User callback function called at the end of a transfer.
<i>userData</i>	User data for callback.
<i>txHandle</i>	DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
<i>rxHandle</i>	DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

### 31.3.5.2 status\_t SPI\_MasterTransferDMA ( **SPI\_Type** \* *base*, **spi\_dma\_handle\_t** \* *handle*, **spi\_transfer\_t** \* *xfer* )

Note

This interface returned immediately after transfer initiates, users should call SPI\_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	SPI DMA handle pointer.
<i>xfer</i>	Pointer to dma transfer structure.

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.
<i>kStatus_SPI_Busy</i>	SPI is not idle, is running another transfer.

### 31.3.5.3 void SPI\_MasterTransferAbortDMA ( **SPI\_Type** \* *base*, **spi\_dma\_handle\_t** \* *handle* )

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	SPI DMA handle pointer.

### 31.3.5.4 status\_t SPI\_MasterTransferGetCountDMA ( **SPI\_Type** \* *base*, **spi\_dma\_handle\_t** \* *handle*, **size\_t** \* *count* )

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	SPI DMA handle pointer.
<i>count</i>	Transferred bytes.

Return values

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

### 31.3.5.5 static void SPI\_SlaveTransferCreateHandleDMA ( **SPI\_Type** \* *base*, **spi\_dma\_handle\_t** \* *handle*, **spi\_dma\_callback\_t** *callback*, **void** \* *userData*, **dma\_handle\_t** \* *txHandle*, **dma\_handle\_t** \* *rxHandle* ) [inline], [static]

This function initializes the SPI slave DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	SPI handle pointer.
<i>callback</i>	User callback function called at the end of a transfer.
<i>userData</i>	User data for callback.
<i>txHandle</i>	DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
<i>rxHandle</i>	DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

### 31.3.5.6 static status\_t SPI\_SlaveTransferDMA ( **SPI\_Type** \* *base*, **spi\_dma\_handle\_t** \* *handle*, **spi\_transfer\_t** \* *xfer* ) [inline], [static]

## Note

This interface returned immediately after transfer initiates, users should call SPI\_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

## Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	SPI DMA handle pointer.
<i>xfer</i>	Pointer to dma transfer structure.

## Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.
<i>kStatus_SPI_Busy</i>	SPI is not idle, is running another transfer.

**31.3.5.7 static void SPI\_SlaveTransferAbortDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle* ) [inline], [static]**

## Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	SPI DMA handle pointer.

**31.3.5.8 static status\_t SPI\_SlaveTransferGetCountDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle*, size\_t \* *count* ) [inline], [static]**

## Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	SPI DMA handle pointer.
<i>count</i>	Transferred bytes.

## Return values

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

## 31.4 SPI FreeRTOS driver

### 31.4.1 Overview

This section describes the programming interface of the SPI FreeRTOS driver.

### Driver version

- `#define FSL_SPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`  
*SPI FreeRTOS driver version.*

### SPI RTOS Operation

- `status_t SPI_RTOS_Init (spi_rtos_handle_t *handle, SPI_Type *base, const spi_master_config_t *masterConfig, uint32_t srcClock_Hz)`  
*Initializes SPI.*
- `status_t SPI_RTOS_Deinit (spi_rtos_handle_t *handle)`  
*Deinitializes the SPI.*
- `status_t SPI_RTOS_Transfer (spi_rtos_handle_t *handle, spi_transfer_t *transfer)`  
*Performs SPI transfer.*

### 31.4.2 Macro Definition Documentation

#### 31.4.2.1 `#define FSL_SPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`

### 31.4.3 Function Documentation

#### 31.4.3.1 `status_t SPI_RTOS_Init ( spi_rtos_handle_t * handle, SPI_Type * base, const spi_master_config_t * masterConfig, uint32_t srcClock_Hz )`

This function initializes the SPI module and related RTOS context.

Parameters

<code>handle</code>	The RTOS SPI handle, the pointer to an allocated space for RTOS context.
<code>base</code>	The pointer base address of the SPI instance to initialize.
<code>masterConfig</code>	Configuration structure to set-up SPI in master mode.

<i>srcClock_Hz</i>	Frequency of input clock of the SPI module.
--------------------	---

Returns

status of the operation.

### 31.4.3.2 status\_t SPI\_RTOS\_Deinit ( *spi\_rtos\_handle\_t \* handle* )

This function deinitializes the SPI module and related RTOS context.

Parameters

<i>handle</i>	The RTOS SPI handle.
---------------	----------------------

### 31.4.3.3 status\_t SPI\_RTOS\_Transfer ( *spi\_rtos\_handle\_t \* handle*, *spi\_transfer\_t \* transfer* )

This function performs an SPI transfer according to data given in the transfer structure.

Parameters

<i>handle</i>	The RTOS SPI handle.
<i>transfer</i>	Structure specifying the transfer parameters.

Returns

status of the operation.

## 31.5 SPI CMSIS driver

This section describes the programming interface of the SPI Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method please refer to <http://www.-keil.com/pack/doc/cmsis/Driver/html/index.html>.

### 31.5.1 Function groups

#### 31.5.1.1 SPI CMSIS GetVersion Operation

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

#### 31.5.1.2 SPI CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

#### 31.5.1.3 SPI 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.

#### 31.5.1.4 SPI CMSIS Transfer Operation

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

#### 31.5.1.5 SPI CMSIS Status Operation

This function group gets the SPI transfer status.

#### 31.5.1.6 SPI CMSIS Control Operation

This function can configure instance as master mode or slave mode, set baudrate for master mode transfer, get current baudrate of master mode transfer, set transfer data bits and other control command.

## 31.5.2 Typical use case

### 31.5.2.1 Master Operation

```
/* Variables */
uint8_t masterRxData[TRANSFER_SIZE] = {0U};
uint8_t masterTxData[TRANSFER_SIZE] = {0U};

/*SPI master init*/
Driver_SPI0.Initialize(SPI_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();
```

### 31.5.2.2 Slave Operation

```
/* Variables */
uint8_t slaveRxData[TRANSFER_SIZE] = {0U};
uint8_t slaveTxData[TRANSFER_SIZE] = {0U};

/*SPI slave init*/
Driver_SPI1.Initialize(SPI_SlaveSignalEvent_t);
Driver_SPI1.PowerControl(ARM_POWER_FULL);
Driver_SPI1.Control(ARM_SPI_MODE_SLAVE, false);

/* Start slave transfer */
Driver_SPI1.Transfer(slaveTxData, slaveRxData, TRANSFER_SIZE);

/* slave power off */
Driver_SPI1.PowerControl(ARM_POWER_OFF);

/* slave uninitialize */
Driver_SPI1.Uninitialize();
```

# Chapter 32

## SYSMPU: System Memory Protection Unit

### 32.1 Overview

The SYSMPU driver provides hardware access control for all memory references generated in the device. Use the SYSMPU driver to program the region descriptors that define memory spaces and their access rights. After initialization, the SYSMPU concurrently monitors the system bus transactions and evaluates their appropriateness.

### 32.2 Initialization and Deinitialization

To initialize the SYSMPU module, call the [SYSMPU\\_Init\(\)](#) function and provide the user configuration data structure. This function sets the configuration of the SYSMPU module automatically and enables the SYSMPU module.

Note that the configuration start address, end address, the region valid value, and the debugger's access permission for the SYSMPU region 0 cannot be changed.

This is an example code to configure the SYSMPU driver.

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

### 32.3 Basic Control Operations

SYSMPU can be enabled/disabled for the entire memory protection region by calling the [SYSMPU\\_Enable\(\)](#) function. To save the power for any unused special regions when the entire memory protection region is disabled, call the [SYSMPU\\_RegionEnable\(\)](#).

After SYSMPU initialization, the `SYSMPU_SetRegionLowMasterAccessRights()` and `SYSMPU_SetRegionHighMasterAccessRights()` can be used to change the access rights for special master ports and for special region numbers. The `SYSMPU_SetRegionConfig` can be used to set the whole region with the start/end address with access rights.

The `SYSMPU_GetHardwareInfo()` API is provided to get the hardware information for the device. The `SYSMPU_GetSlavePortErrorStatus()` API is provided to get the error status of a special slave port. When an error happens in this port, the `SYSMPU_GetDetailErrorAccessInfo()` API is provided to get the detailed error information.

### Data Structures

- struct `sysmpu_hardware_info_t`  
*SYSMPU hardware basic information.* [More...](#)
- struct `sysmpu_access_err_info_t`  
*SYSMPU detail error access information.* [More...](#)
- struct `sysmpu_rwxrights_master_access_control_t`

*SYSMPU read/write/execute rights control for bus master 0 ~ 3. [More...](#)*

- struct `sysmpu_rwrights_master_access_control_t`  
*SYSMPU read/write access control for bus master 4 ~ 7. [More...](#)*

- struct `sysmpu_region_config_t`  
*SYSMPU region configuration structure. [More...](#)*

- struct `sysmpu_config_t`  
*The configuration structure for the SYSMPU initialization. [More...](#)*

## Macros

- `#define SYSMPU_MASTER_RWATTRIBUTE_START_PORT (4U)`  
*define the start master port with read and write attributes.*
- `#define SYSMPU_REGION_RWXRIGHTS_MASTER_SHIFT(n) ((n)*6U)`  
*SYSMPU the bit shift for masters with privilege rights: read write and execute.*
- `#define SYSMPU_REGION_RWXRIGHTS_MASTER_MASK(n) (0x1FUL << SYSMPU_REGION_RWXRIGHTS_MASTER_SHIFT(n))`  
*SYSMPU masters with read, write and execute rights bit mask.*
- `#define SYSMPU_REGION_RWXRIGHTS_MASTER_WIDTH 5U`  
*SYSMPU masters with read, write and execute rights bit width.*
- `#define SYSMPU_REGION_RWXRIGHTS_MASTER(n, x) (((uint32_t)((uint32_t)(x)) << SYSMPU_REGION_RWXRIGHTS_MASTER_SHIFT(n))) & SYSMPU_REGION_RWXRIGHTS_MASTER_MASK(n))`  
*SYSMPU masters with read, write and execute rights priority setting.*
- `#define SYSMPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT(n) ((n)*6U + SYSMPU_REGION_RWXRIGHTS_MASTER_WIDTH)`  
*SYSMPU masters with read, write and execute rights process enable bit shift.*
- `#define SYSMPU_REGION_RWXRIGHTS_MASTER_PE_MASK(n) (0x1UL << SYSMPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT(n))`  
*SYSMPU masters with read, write and execute rights process enable bit mask.*
- `#define SYSMPU_REGION_RWXRIGHTS_MASTER_PE(n, x)`  
*SYSMPU masters with read, write and execute rights process enable setting.*
- `#define SYSMPU_REGION_RWRIGHTS_MASTER_SHIFT(n) (((n)-SYSMPU_MASTER_RW_ATTRIBUTE_START_PORT) * 2U + 24U)`  
*SYSMPU masters with normal read write permission bit shift.*
- `#define SYSMPU_REGION_RWRIGHTS_MASTER_MASK(n) (0x3UL << SYSMPU_REGION_RWRIGHTS_MASTER_SHIFT(n))`  
*SYSMPU masters with normal read write rights bit mask.*
- `#define SYSMPU_REGION_RWRIGHTS_MASTER(n, x) (((uint32_t)((uint32_t)(x)) << SYSMPU_REGION_RWRIGHTS_MASTER_SHIFT(n))) & SYSMPU_REGION_RWRIGHTS_MASTER_MASK(n))`  
*SYSMPU masters with normal read write rights priority setting.*

## Enumerations

- enum `sysmpu_region_total_num_t` {
   
  `kSYSMPU_8Regions` = 0x0U,
   
  `kSYSMPU_12Regions` = 0x1U,
   
  `kSYSMPU_16Regions` = 0x2U
 }

*Describes the number of SYSMPU regions.*

- enum `sysmpu_slave_t` {
   
  `kSYSMPU_Slave0` = 0U,
   
  `kSYSMPU_Slave1` = 1U,
   
  `kSYSMPU_Slave2` = 2U,
   
  `kSYSMPU_Slave3` = 3U,
   
  `kSYSMPU_Slave4` = 4U }
   
    *SYSMPU slave port number.*
- enum `sysmpu_err_access_control_t` {
   
  `kSYSMPU_NoRegionHit` = 0U,
   
  `kSYSMPU_NoneOverlappRegion` = 1U,
   
  `kSYSMPU_OverlappRegion` = 2U }
   
    *SYSMPU error access control detail.*
- enum `sysmpu_err_access_type_t` {
   
  `kSYSMPU_ErrTypeRead` = 0U,
   
  `kSYSMPU_ErrTypeWrite` = 1U }
   
    *SYSMPU error access type.*
- enum `sysmpu_err_attributes_t` {
   
  `kSYSMPU_InstructionAccessInUserMode` = 0U,
   
  `kSYSMPU_DataAccessInUserMode` = 1U,
   
  `kSYSMPU_InstructionAccessInSupervisorMode` = 2U,
   
  `kSYSMPU_DataAccessInSupervisorMode` = 3U }
   
    *SYSMPU access error attributes.*
- enum `sysmpu_supervisor_access_rights_t` {
   
  `kSYSMPU_SupervisorReadWriteExecute` = 0U,
   
  `kSYSMPU_SupervisorReadExecute` = 1U,
   
  `kSYSMPU_SupervisorReadWrite` = 2U,
   
  `kSYSMPU_SupervisorEqualToUsermode` = 3U }
   
    *SYSMPU access rights in supervisor mode for bus master 0 ~ 3.*
- enum `sysmpu_user_access_rights_t` {
   
  `kSYSMPU_UserNoAccessRights` = 0U,
   
  `kSYSMPU_UserExecute` = 1U,
   
  `kSYSMPU_UserWrite` = 2U,
   
  `kSYSMPU_UserWriteExecute` = 3U,
   
  `kSYSMPU_UserRead` = 4U,
   
  `kSYSMPU_UserReadExecute` = 5U,
   
  `kSYSMPU_UserReadWrite` = 6U,
   
  `kSYSMPU_UserReadWriteExecute` = 7U }
   
    *SYSMPU access rights in user mode for bus master 0 ~ 3.*

## Driver version

- #define `FSL_SYSMPU_DRIVER_VERSION` (`MAKE_VERSION(2, 2, 3)`)
   
    *SYSMPU driver version 2.2.3.*

## Initialization and deinitialization

- void `SYSMPU_Init` (`SYSMPU_Type` \*base, const `sysmpu_config_t` \*config)

- **`SYSMPU_Deinit`** (`SYSMPU_Type` \*base)
 

*Initializes the SYSMPU with the user configuration structure.*

*Deinitializes the SYSMPU regions.*

## Basic Control Operations

- static void **`SYSMPU_Enable`** (`SYSMPU_Type` \*base, bool enable)
 

*Enables/disables the SYSMPU globally.*
- static void **`SYSMPU_RegionEnable`** (`SYSMPU_Type` \*base, `uint32_t` number, bool enable)
 

*Enables/disables the SYSMPU for a special region.*
- void **`SYSMPU_GetHardwareInfo`** (`SYSMPU_Type` \*base, `sysmpu_hardware_info_t` \*hardwareInform)
 

*Gets the SYSMPU basic hardware information.*
- void **`SYSMPU_SetRegionConfig`** (`SYSMPU_Type` \*base, const `sysmpu_region_config_t` \*regionConfig)
 

*Sets the SYSMPU region.*
- void **`SYSMPU_SetRegionAddr`** (`SYSMPU_Type` \*base, `uint32_t` regionNum, `uint32_t` startAddr, `uint32_t` endAddr)
 

*Sets the region start and end address.*
- void **`SYSMPU_SetRegionRwxMasterAccessRights`** (`SYSMPU_Type` \*base, `uint32_t` regionNum, `uint32_t` masterNum, const `sysmpu_rwxrights_master_access_control_t` \*accessRights)
 

*Sets the SYSMPU region access rights for masters with read, write, and execute rights.*
- bool **`SYSMPU_GetSlavePortErrorStatus`** (`SYSMPU_Type` \*base, `sysmpu_slave_t` slaveNum)
 

*Gets the numbers of slave ports where errors occur.*
- void **`SYSMPU_GetDetailErrorAccessInfo`** (`SYSMPU_Type` \*base, `sysmpu_slave_t` slaveNum, `sysmpu_access_err_info_t` \*errInform)
 

*Gets the SYSMPU detailed error access information.*

## 32.4 Data Structure Documentation

### 32.4.1 struct `sysmpu_hardware_info_t`

#### Data Fields

- `uint8_t hardwareRevisionLevel`

*Specifies the SYSMPU's hardware and definition reversion level.*
- `uint8_t slavePortsNumbers`

*Specifies the number of slave ports connected to SYSMPU.*
- `sysmpu_region_total_num_t regionsNumbers`

*Indicates the number of region descriptors implemented.*

#### Field Documentation

- (1) `uint8_t sysmpu_hardware_info_t::hardwareRevisionLevel`
- (2) `uint8_t sysmpu_hardware_info_t::slavePortsNumbers`
- (3) `sysmpu_region_total_num_t sysmpu_hardware_info_t::regionsNumbers`

### 32.4.2 struct sysmpu\_access\_err\_info\_t

#### Data Fields

- `uint32_t master`  
*Access error master.*
- `sysmpu_err_attributes_t attributes`  
*Access error attributes.*
- `sysmpu_err_access_type_t accessType`  
*Access error type.*
- `sysmpu_err_access_control_t accessControl`  
*Access error control.*
- `uint32_t address`  
*Access error address.*
- `uint8_t processorIdentification`  
*Access error processor identification.*

#### Field Documentation

- (1) `uint32_t sysmpu_access_err_info_t::master`
- (2) `sysmpu_err_attributes_t sysmpu_access_err_info_t::attributes`
- (3) `sysmpu_err_access_type_t sysmpu_access_err_info_t::accessType`
- (4) `sysmpu_err_access_control_t sysmpu_access_err_info_t::accessControl`
- (5) `uint32_t sysmpu_access_err_info_t::address`
- (6) `uint8_t sysmpu_access_err_info_t::processorIdentification`

### 32.4.3 struct sysmpu\_rwxrights\_master\_access\_control\_t

#### Data Fields

- `sysmpu_supervisor_access_rights_t superAccessRights`  
*Master access rights in supervisor mode.*
- `sysmpu_user_access_rights_t userAccessRights`  
*Master access rights in user mode.*
- `bool processIdentifierEnable`  
*Enables or disables process identifier.*

#### Field Documentation

- (1) `sysmpu_supervisor_access_rights_t sysmpu_rwxrights_master_access_control_t::superAccessRights`

(2) `sysmpu_user_access_rights_t sysmpu_rwxrights_master_access_control_t::userAccessRights`

(3) `bool sysmpu_rwxrights_master_access_control_t::processIdentifierEnable`

### 32.4.4 struct sysmpu\_rwrights\_master\_access\_control\_t

#### Data Fields

- `bool writeEnable`  
*Enables or disables write permission.*
- `bool readEnable`  
*Enables or disables read permission.*

#### Field Documentation

(1) `bool sysmpu_rwrights_master_access_control_t::writeEnable`

(2) `bool sysmpu_rwrights_master_access_control_t::readEnable`

### 32.4.5 struct sysmpu\_region\_config\_t

This structure is used to configure the regionNum region. The accessRights1[0] ~ accessRights1[3] are used to configure the bus master 0 ~ 3 with the privilege rights setting. The accessRights2[0] ~ accessRights2[3] are used to configure the high master 4 ~ 7 with the normal read write permission. The master port assignment is the chip configuration. Normally, the core is the master 0, debugger is the master 1. Note that the SY SMPU assigns a priority scheme where the debugger is treated as the highest priority master followed by the core and then all the remaining masters. SY SMPU protection does not allow writes from the core to affect the "regionNum 0" start and end address nor the permissions associated with the debugger. It can only write the permission fields associated with the other masters. This protection guarantees that the debugger always has access to the entire address space and those rights can't be changed by the core or any other bus master. Prepare the region configuration when regionNum is 0.

#### Data Fields

- `uint32_t regionNum`  
*SY SMPU region number, range form 0 ~ FSL\_FEATURE\_SYSMPU\_DESCRIPTOR\_COUNT - 1.*
- `uint32_t startAddress`  
*Memory region start address.*
- `uint32_t endAddress`  
*Memory region end address.*
- `sysmpu_rwxrights_master_access_control_t accessRights1 [4]`  
*Masters with read, write and execute rights setting.*
- `sysmpu_rwrights_master_access_control_t accessRights2 [4]`  
*Masters with normal read write rights setting.*
- `uint8_t processIdentifier`

- `uint8_t processIdMask`  
*Process identifier mask.*

#### Field Documentation

- (1) `uint32_t sysmpu_region_config_t::regionNum`
- (2) `uint32_t sysmpu_region_config_t::startAddress`

Note: bit0 ~ bit4 always be marked as 0 by SY SMPU. The actual start address is 0-modulo-32 byte address.

- (3) `uint32_t sysmpu_region_config_t::endAddress`

Note: bit0 ~ bit4 always be marked as 1 by SY SMPU. The actual end address is 31-modulo-32 byte address.

- (4) `sysmpu_rwxrights_master_access_control_t sysmpu_region_config_t::accessRights1[4]`
- (5) `sysmpu_rwrights_master_access_control_t sysmpu_region_config_t::accessRights2[4]`
- (6) `uint8_t sysmpu_region_config_t::processIdentifier`
- (7) `uint8_t sysmpu_region_config_t::processIdMask`

The setting bit will ignore the same bit in process identifier.

### 32.4.6 struct sysmpu\_config\_t

This structure is used when calling the SY SMPU\_Init function.

#### Data Fields

- `sysmpu_region_config_t regionConfig`  
*Region access permission.*
- `struct _sysmpu_config * next`  
*Pointer to the next structure.*

#### Field Documentation

- (1) `sysmpu_region_config_t sysmpu_config_t::regionConfig`
- (2) `struct _sysmpu_config* sysmpu_config_t::next`

## 32.5 Macro Definition Documentation

- 32.5.1 #define FSL\_SYSMPU\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 3))
- 32.5.2 #define SYSMPU\_MASTER\_RWATTRIBUTE\_START\_PORT (4U)
- 32.5.3 #define SYSMPU\_REGION\_RWXRIGHTS\_MASTER\_SHIFT( *n* ) ((*n*)\*6U)
- 32.5.4 #define SYSMPU\_REGION\_RWXRIGHTS\_MASTER\_MASK( *n* ) (0x1FUL << SYSMPU\_REGION\_RWXRIGHTS\_MASTER\_SHIFT(*n*))
- 32.5.5 #define SYSMPU\_REGION\_RWXRIGHTS\_MASTER\_WIDTH 5U
- 32.5.6 #define SYSMPU\_REGION\_RWXRIGHTS\_MASTER( *n*, *x* ) (((uint32\_t)((uint32\_t)(*x*) << SYSMPU\_REGION\_RWXRIGHTS\_MASTER\_SHIFT(*n*))) & SYSMPU\_REGION\_RWXRIGHTS\_MASTER\_MASK(*n*))
- 32.5.7 #define SYSMPU\_REGION\_RWXRIGHTS\_MASTER\_PE\_SHIFT( *n* ) ((*n*)\*6U + SYSMPU\_REGION\_RWXRIGHTS\_MASTER\_WIDTH)
- 32.5.8 #define SYSMPU\_REGION\_RWXRIGHTS\_MASTER\_PE\_MASK( *n* ) (0x1UL << SYSMPU\_REGION\_RWXRIGHTS\_MASTER\_PE\_SHIFT(*n*))
- 32.5.9 #define SYSMPU\_REGION\_RWXRIGHTS\_MASTER\_PE( *n*, *x* )

**Value:**

```
((((uint32_t) (((uint32_t) (x) << SYSMPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT
(n))) & \
SYSMPU_REGION_RWXRIGHTS_MASTER_PE_MASK(n))
```

- 32.5.10 #define SYSMPU\_REGION\_RWRIGHTS\_MASTER\_SHIFT( *n* ) (((*n*)-SYSMPU\_MASTER\_RWATTRIBUTE\_START\_PORT) \* 2U + 24U)
- 32.5.11 #define SYSMPU\_REGION\_RWRIGHTS\_MASTER\_MASK( *n* ) (0x3UL << SYSMPU\_REGION\_RWRIGHTS\_MASTER\_SHIFT(*n*))
- 32.5.12 #define SYSMPU\_REGION\_RWRIGHTS\_MASTER( *n*, *x* ) (((uint32\_t)((uint32\_t)(*x*) << SYSMPU\_REGION\_RWRIGHTS\_MASTER\_SHIFT(*n*))) & SYSMPU\_REGION\_RWRIGHTS\_MASTER\_MASK(*n*))

## 32.6 Enumeration Type Documentation

### 32.6.1 enum sysmpu\_region\_total\_num\_t

Enumerator

- kSYSMPU\_8Regions* SYSMPU supports 8 regions.
- kSYSMPU\_12Regions* SYSMPU supports 12 regions.
- kSYSMPU\_16Regions* SYSMPU supports 16 regions.

### 32.6.2 enum sysmpu\_slave\_t

Enumerator

- kSYSMPU\_Slave0* SYSMPU slave port 0.
- kSYSMPU\_Slave1* SYSMPU slave port 1.
- kSYSMPU\_Slave2* SYSMPU slave port 2.
- kSYSMPU\_Slave3* SYSMPU slave port 3.
- kSYSMPU\_Slave4* SYSMPU slave port 4.

### 32.6.3 enum sysmpu\_err\_access\_control\_t

Enumerator

- kSYSMPU\_NoRegionHit* No region hit error.
- kSYSMPU\_NoneOverlapRegion* Access single region error.
- kSYSMPU\_OverlapRegion* Access overlapping region error.

### 32.6.4 enum sysmpu\_err\_access\_type\_t

Enumerator

- kSYSMPU\_ErrTypeRead* SYSMPU error access type — read.
- kSYSMPU\_ErrTypeWrite* SYSMPU error access type — write.

### 32.6.5 enum sysmpu\_err\_attributes\_t

Enumerator

- kSYSMPU\_InstructionAccessInUserMode* Access instruction error in user mode.
- kSYSMPU\_DataAccessInUserMode* Access data error in user mode.
- kSYSMPU\_InstructionAccessInSupervisorMode* Access instruction error in supervisor mode.
- kSYSMPU\_DataAccessInSupervisorMode* Access data error in supervisor mode.

### 32.6.6 enum sysmpu\_supervisor\_access\_rights\_t

Enumerator

***kSYSMPU\_SupervisorReadWriteExecute*** Read write and execute operations are allowed in supervisor mode.

***kSYSMPU\_SupervisorReadExecute*** Read and execute operations are allowed in supervisor mode.

***kSYSMPU\_SupervisorReadWrite*** Read write operations are allowed in supervisor mode.

***kSYSMPU\_SupervisorEqualToUsermode*** Access permission equal to user mode.

### 32.6.7 enum sysmpu\_user\_access\_rights\_t

Enumerator

***kSYSMPU\_UserNoAccessRights*** No access allowed in user mode.

***kSYSMPU\_UserExecute*** Execute operation is allowed in user mode.

***kSYSMPU\_UserWrite*** Write operation is allowed in user mode.

***kSYSMPU\_UserWriteExecute*** Write and execute operations are allowed in user mode.

***kSYSMPU\_UserRead*** Read is allowed in user mode.

***kSYSMPU\_UserReadExecute*** Read and execute operations are allowed in user mode.

***kSYSMPU\_UserReadWrite*** Read and write operations are allowed in user mode.

***kSYSMPU\_UserReadWriteExecute*** Read write and execute operations are allowed in user mode.

## 32.7 Function Documentation

### 32.7.1 void SYSMPU\_Init ( **SYSMPU\_Type** \* *base*, **const sysmpu\_config\_t** \* *config* )

This function configures the SYSMPU module with the user-defined configuration.

Parameters

<i>base</i>	SYSMPU peripheral base address.
<i>config</i>	The pointer to the configuration structure.

### 32.7.2 void SYSMPU\_Deinit ( **SYSMPU\_Type** \* *base* )

Parameters

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

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

Call this API to enable or disable the SYSSMPU module.

Parameters

<i>base</i>	SYSSMPU peripheral base address.
<i>enable</i>	True enable SYSSMPU, false disable SYSSMPU.

### 32.7.4 static void SYSSMPU\_RegionEnable ( **SYSSMPU\_Type** \* *base*, **uint32\_t number**, **bool enable** ) [inline], [static]

When SYSSMPU is enabled, call this API to disable an unused region of an enabled SYSSMPU. Call this API to minimize the power dissipation.

Parameters

<i>base</i>	SYSSMPU peripheral base address.
<i>number</i>	SYSSMPU region number.
<i>enable</i>	True enable the special region SYSSMPU, false disable the special region SYSSMPU.

### 32.7.5 void SYSSMPU\_GetHardwareInfo ( **SYSSMPU\_Type** \* *base*, **sysmpu.hardware\_info\_t** \* *hardwareInform* )

Parameters

<i>base</i>	SYSSMPU peripheral base address.
<i>hardware- Inform</i>	The pointer to the SYSSMPU hardware information structure. See "sysmpu.hardware- _info_t".

### 32.7.6 void SYSMPU\_SetRegionConfig ( **SYSMPU\_Type** \* *base*, const **sysmpu\_region\_config\_t** \* *regionConfig* )

Note: Due to the SYSMPU protection, the region number 0 does not allow writes from core to affect the start and end address nor the permissions associated with the debugger. It can only write the permission fields associated with the other masters.

Parameters

<i>base</i>	SYSMPU peripheral base address.
<i>regionConfig</i>	The pointer to the SYSMPU user configuration structure. See "sysmpu_region_config_t".

### 32.7.7 void SYSMPU\_SetRegionAddr ( **SYSMPU\_Type** \* *base*, **uint32\_t** *regionNum*, **uint32\_t** *startAddr*, **uint32\_t** *endAddr* )

Memory region start address. Note: bit0 ~ bit4 is always marked as 0 by SYSMPU. The actual start address by SYSMPU is 0-modulo-32 byte address. Memory region end address. Note: bit0 ~ bit4 always be marked as 1 by SYSMPU. The end address used by the SYSMPU is 31-modulo-32 byte address. Note: Due to the SYSMPU protection, the startAddr and endAddr can't be changed by the core when regionNum is 0.

Parameters

<i>base</i>	SYSMPU peripheral base address.
<i>regionNum</i>	SYSMPU region number. The range is from 0 to FSL FEATURE_SYSMPU_DESCRIPTOR_COUNT - 1.
<i>startAddr</i>	Region start address.
<i>endAddr</i>	Region end address.

### 32.7.8 void SYSMPU\_SetRegionRwxMasterAccessRights ( **SYSMPU\_Type** \* *base*, **uint32\_t** *regionNum*, **uint32\_t** *masterNum*, const **sysmpu\_rwxrights\_master\_access\_control\_t** \* *accessRights* )

The SYSMPU access rights depend on two board classifications of bus masters. The privilege rights masters and the normal rights masters. The privilege rights masters have the read, write, and execute access rights. Except the normal read and write rights, the execute rights are also allowed for these masters. The privilege rights masters normally range from bus masters 0 - 3. However, the maximum master number is device-specific. See the "SYSMPU\_PRIVILEGED\_RIGHTS\_MASTER\_MAX\_INDEX". The normal rights masters access rights control see "SYSMPU\_SetRegionRwMasterAccessRights()".

Parameters

<i>base</i>	SYSSMPU peripheral base address.
<i>regionNum</i>	SYSSMPU region number. Should range from 0 to FSL_FEATURE_SYSMPU_DESCRIPTOR_COUNT - 1.
<i>masterNum</i>	SYSSMPU bus master number. Should range from 0 to SYSMPU_PRIVILEGED_RIGHTS_MASTER_MAX_INDEX.
<i>accessRights</i>	The pointer to the SYSSMPU access rights configuration. See "sysmpu_rwxrights_master_access_control_t".

### 32.7.9 **bool SYSMPU\_GetSlavePortErrorStatus ( SYSMPU\_Type \* *base*, sysmpu\_slave\_t *slaveNum* )**

Parameters

<i>base</i>	SYSSMPU peripheral base address.
<i>slaveNum</i>	SYSSMPU slave port number.

Returns

The slave ports error status. true - error happens in this slave port. false - error didn't happen in this slave port.

### 32.7.10 **void SYSMPU\_GetDetailErrorAccessInfo ( SYSMPU\_Type \* *base*, sysmpu\_slave\_t *slaveNum*, sysmpu\_access\_err\_info\_t \* *errInform* )**

Parameters

<i>base</i>	SYSSMPU peripheral base address.
<i>slaveNum</i>	SYSSMPU slave port number.
<i>errInform</i>	The pointer to the SYSSMPU access error information. See "sysmpu_access_err_info_t".

# Chapter 33

## UART: Universal Asynchronous Receiver/Transmitter Driver

### 33.1 Overview

#### Modules

- [UART CMSIS Driver](#)
- [UART DMA Driver](#)
- [UART Driver](#)
- [UART FreeRTOS Driver](#)

## 33.2 UART Driver

### 33.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Universal Asynchronous Receiver/Transmitter (UART) module of MCUXpresso SDK devices.

The UART driver includes functional APIs and transactional APIs.

Functional APIs are used for UART initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the UART peripheral and how to organize functional APIs to meet the application requirements. All functional APIs use the peripheral base address as the first parameter. UART functional operation groups provide the functional API set.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the `uart_handle_t` as the second parameter. Initialize the handle by calling the [UART\\_TransferCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer, which means that the functions [UART\\_TransferSendNonBlocking\(\)](#) and [UART\\_TransferReceiveNonBlocking\(\)](#) set up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_UART_TxIdle` and `kStatus_UART_RxIdle`.

Transactional receive APIs support the ring buffer. Prepare the memory for the ring buffer and pass in the start address and size while calling the [UART\\_TransferCreateHandle\(\)](#). If passing NULL, the ring buffer feature is disabled. When the ring buffer is enabled, the received data is saved to the ring buffer in the background. The [UART\\_TransferReceiveNonBlocking\(\)](#) function first gets data from the ring buffer. If the ring buffer does not have enough data, the function first returns the data in the ring buffer and then saves the received data to user memory. When all data is received, the upper layer is informed through a callback with the `kStatus_UART_RxIdle`.

If the receive ring buffer is full, the upper layer is informed through a callback with the `kStatus_UART_RxRingBufferOverrun`. In the callback function, the upper layer reads data out from the ring buffer. If not, existing data is overwritten by the new data.

The ring buffer size is specified when creating the handle. Note that one byte is reserved for the ring buffer maintenance. When creating handle using the following code.

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/uart`. In this example, the buffer size is 32, but only 31 bytes are used for saving data.

### 33.2.2 Typical use case

#### 33.2.2.1 UART Send/receive using a polling method

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

### 33.2.2.2 UART Send/receive using an interrupt method

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

### 33.2.2.3 UART Receive using the ringbuffer feature

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

### 33.2.2.4 UART Send/Receive using the DMA method

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

## Data Structures

- struct [uart\\_config\\_t](#)  
*UART configuration structure.* [More...](#)
- struct [uart\\_transfer\\_t](#)  
*UART transfer structure.* [More...](#)
- struct [uart\\_handle\\_t](#)  
*UART handle structure.* [More...](#)

## Macros

- #define [UART\\_RETRY\\_TIMES](#) 0U /\* Defining to zero means to keep waiting for the flag until it is assert/deassert. \*/  
*Retry times for waiting flag.*

## TypeDefs

- typedef void(\* [uart\\_transfer\\_callback\\_t](#) )(UART\_Type \*base, uart\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*UART transfer callback function.*

## Enumerations

- enum {
   
kStatus\_UART\_TxBusy = MAKE\_STATUS(kStatusGroup\_UART, 0),
   
kStatus\_UART\_RxBusy = MAKE\_STATUS(kStatusGroup\_UART, 1),
   
kStatus\_UART\_TxIdle = MAKE\_STATUS(kStatusGroup\_UART, 2),
   
kStatus\_UART\_RxIdle = MAKE\_STATUS(kStatusGroup\_UART, 3),
   
kStatus\_UART\_TxWatermarkTooLarge = MAKE\_STATUS(kStatusGroup\_UART, 4),
   
kStatus\_UART\_RxWatermarkTooLarge = MAKE\_STATUS(kStatusGroup\_UART, 5),
   
kStatus\_UART\_FlagCannotClearManually,
   
kStatus\_UART\_Error = MAKE\_STATUS(kStatusGroup\_UART, 7),
   
kStatus\_UART\_RxRingBufferOverrun = MAKE\_STATUS(kStatusGroup\_UART, 8),
   
kStatus\_UART\_RxHardwareOverrun = MAKE\_STATUS(kStatusGroup\_UART, 9),
   
kStatus\_UART\_NoiseError = MAKE\_STATUS(kStatusGroup\_UART, 10),
   
kStatus\_UART\_FramingError = MAKE\_STATUS(kStatusGroup\_UART, 11),
   
kStatus\_UART\_ParityError = MAKE\_STATUS(kStatusGroup\_UART, 12),
   
kStatus\_UART\_BaudrateNotSupport,
   
kStatus\_UART\_IdleLineDetected = MAKE\_STATUS(kStatusGroup\_UART, 14),
   
kStatus\_UART\_Timeout = MAKE\_STATUS(kStatusGroup\_UART, 15) }

*Error codes for the UART driver.*

- enum `uart_parity_mode_t` {
   
kUART\_ParityDisabled = 0x0U,
   
kUART\_ParityEven = 0x2U,
   
kUART\_ParityOdd = 0x3U }
- UART parity mode.*
- enum `uart_stop_bit_count_t` {
   
kUART\_OneStopBit = 0U,
   
kUART\_TwoStopBit = 1U }
- UART stop bit count.*
- enum `uart_idle_type_select_t` {
   
kUART\_IdleTypeStartBit = 0U,
   
kUART\_IdleTypeStopBit = 1U }
- UART idle type select.*
- enum `_uart_interrupt_enable` {
   
kUART\_RxActiveEdgeInterruptEnable = (UART\_BDH\_RXEDGIE\_MASK),
   
kUART\_TxDataRegEmptyInterruptEnable = (UART\_C2\_TIE\_MASK << 8),
   
kUART\_TransmissionCompleteInterruptEnable = (UART\_C2\_TCIE\_MASK << 8),
   
kUART\_RxDataRegFullInterruptEnable = (UART\_C2\_RIE\_MASK << 8),
   
kUART\_IdleLineInterruptEnable = (UART\_C2\_ILIE\_MASK << 8),
   
kUART\_RxOverrunInterruptEnable = (UART\_C3\_ORIE\_MASK << 16),
   
kUART\_NoiseErrorInterruptEnable = (UART\_C3\_NEIE\_MASK << 16),
   
kUART\_FramingErrorInterruptEnable = (UART\_C3\_FEIE\_MASK << 16),
   
kUART\_ParityErrorInterruptEnable = (UART\_C3\_PEIE\_MASK << 16),
   
kUART\_RxFifoOverflowInterruptEnable = (UART\_CFIFO\_RXOFE\_MASK << 24),
   
kUART\_TxFifoOverflowInterruptEnable = (UART\_CFIFO\_TXOFE\_MASK << 24),
   
kUART\_RxFifoUnderflowInterruptEnable = (UART\_CFIFO\_RXUFE\_MASK << 24) }

- UART interrupt configuration structure, default settings all disabled.*
- enum {
   
kUART\_TxDataRegEmptyFlag = (UART\_S1\_TDRE\_MASK),  
 kUART\_TransmissionCompleteFlag = (UART\_S1\_TC\_MASK),  
 kUART\_RxDataRegFullFlag = (UART\_S1\_RDRF\_MASK),  
 kUART\_IdleLineFlag = (UART\_S1\_IDLE\_MASK),  
 kUART\_RxOverrunFlag = (UART\_S1\_OR\_MASK),  
 kUART\_NoiseErrorFlag = (UART\_S1\_NF\_MASK),  
 kUART\_FramingErrorFlag = (UART\_S1\_FE\_MASK),  
 kUART\_ParityErrorFlag = (UART\_S1\_PF\_MASK),  
 kUART\_RxActiveEdgeFlag,  
 kUART\_RxActiveFlag,  
 kUART\_NoiseErrorInRxDataRegFlag = (UART\_ED\_NOISY\_MASK << 16),  
 kUART\_ParityErrorInRxDataRegFlag = (UART\_ED\_PARITYE\_MASK << 16),  
 kUART\_TxFifoEmptyFlag = (int)(UART\_SFIFO\_TXEMPT\_MASK << 24),  
 kUART\_RxFifoEmptyFlag = (UART\_SFIFO\_RXEMPT\_MASK << 24),  
 kUART\_TxFifoOverflowFlag = (UART\_SFIFO\_TXOF\_MASK << 24),  
 kUART\_RxFifoOverflowFlag = (UART\_SFIFO\_RXOF\_MASK << 24),  
 kUART\_RxFifoUnderflowFlag = (UART\_SFIFO\_RXUF\_MASK << 24) }
- UART status flags.*

## Functions

- `uint32_t UART_GetInstance (UART_Type *base)`  
*Get the UART instance from peripheral base address.*

## Variables

- `void * s_uartHandle []`  
*Pointers to uart handles for each instance.*
- `uart_isr_t s_uartIsr`  
*Pointer to uart IRQ handler for each instance.*

## Driver version

- `#define FSL_UART_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))`  
*UART driver version.*

## Initialization and deinitialization

- `status_t UART_Init (UART_Type *base, const uart_config_t *config, uint32_t srcClock_Hz)`  
*Initializes a UART instance with a user configuration structure and peripheral clock.*
- `void UART_Deinit (UART_Type *base)`

- Deinitializes a UART instance.
- void **UART\_GetDefaultConfig** (uart\_config\_t \*config)  
Gets the default configuration structure.
- status\_t **UART\_SetBaudRate** (UART\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)  
Sets the UART instance baud rate.
- void **UART\_Enable9bitMode** (UART\_Type \*base, bool enable)  
Enable 9-bit data mode for UART.
- static void **UART\_SetMatchAddress** (UART\_Type \*base, uint8\_t address1, uint8\_t address2)  
Set the UART slave address.
- static void **UART\_EnableMatchAddress** (UART\_Type \*base, bool match1, bool match2)  
Enable the UART match address feature.
- static void **UART\_Set9thTransmitBit** (UART\_Type \*base)  
Set UART 9th transmit bit.
- static void **UART\_Clear9thTransmitBit** (UART\_Type \*base)  
Clear UART 9th transmit bit.

## Status

- uint32\_t **UART\_GetStatusFlags** (UART\_Type \*base)  
Gets UART status flags.
- status\_t **UART\_ClearStatusFlags** (UART\_Type \*base, uint32\_t mask)  
Clears status flags with the provided mask.

## Interrupts

- void **UART\_EnableInterrupts** (UART\_Type \*base, uint32\_t mask)  
Enables UART interrupts according to the provided mask.
- void **UART\_DisableInterrupts** (UART\_Type \*base, uint32\_t mask)  
Disables the UART interrupts according to the provided mask.
- uint32\_t **UART\_GetEnabledInterrupts** (UART\_Type \*base)  
Gets the enabled UART interrupts.

## DMA Control

- static uint32\_t **UART\_GetDataRegisterAddress** (UART\_Type \*base)  
Gets the UART data register address.
- static void **UART\_EnableTxDMA** (UART\_Type \*base, bool enable)  
Enables or disables the UART transmitter DMA request.
- static void **UART\_EnableRxDMA** (UART\_Type \*base, bool enable)  
Enables or disables the UART receiver DMA.

## Bus Operations

- static void **UART\_EnableTx** (UART\_Type \*base, bool enable)  
Enables or disables the UART transmitter.
- static void **UART\_EnableRx** (UART\_Type \*base, bool enable)

*Enables or disables the UART receiver.*

- static void [UART\\_WriteByte](#) (UART\_Type \*base, uint8\_t data)  
*Writes to the TX register.*
- static uint8\_t [UART\\_ReadByte](#) (UART\_Type \*base)  
*Reads the RX register directly.*
- static uint8\_t [UART\\_GetRxFifoCount](#) (UART\_Type \*base)  
*Gets the rx FIFO data count.*
- static uint8\_t [UART\\_GetTxFifoCount](#) (UART\_Type \*base)  
*Gets the tx FIFO data count.*
- void [UART\\_SendAddress](#) (UART\_Type \*base, uint8\_t address)  
*Transmit an address frame in 9-bit data mode.*
- status\_t [UART\\_WriteBlocking](#) (UART\_Type \*base, const uint8\_t \*data, size\_t length)  
*Writes to the TX register using a blocking method.*
- status\_t [UART\\_ReadBlocking](#) (UART\_Type \*base, uint8\_t \*data, size\_t length)  
*Read RX data register using a blocking method.*

## Transactional

- void [UART\\_TransferCreateHandle](#) (UART\_Type \*base, uart\_handle\_t \*handle, [uart\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the UART handle.*
- void [UART\\_TransferStartRingBuffer](#) (UART\_Type \*base, uart\_handle\_t \*handle, uint8\_t \*ringBuffer, size\_t ringBufferSize)  
*Sets up the RX ring buffer.*
- void [UART\\_TransferStopRingBuffer](#) (UART\_Type \*base, uart\_handle\_t \*handle)  
*Aborts the background transfer and uninstalls the ring buffer.*
- size\_t [UART\\_TransferGetRxRingBufferLength](#) (uart\_handle\_t \*handle)  
*Get the length of received data in RX ring buffer.*
- status\_t [UART\\_TransferSendNonBlocking](#) (UART\_Type \*base, uart\_handle\_t \*handle, [uart\\_transfer\\_t](#) \*xfer)  
*Transmits a buffer of data using the interrupt method.*
- void [UART\\_TransferAbortSend](#) (UART\_Type \*base, uart\_handle\_t \*handle)  
*Aborts the interrupt-driven data transmit.*
- status\_t [UART\\_TransferGetSendCount](#) (UART\_Type \*base, uart\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of bytes sent out to bus.*
- status\_t [UART\\_TransferReceiveNonBlocking](#) (UART\_Type \*base, uart\_handle\_t \*handle, [uart\\_transfer\\_t](#) \*xfer, size\_t \*receivedBytes)  
*Receives a buffer of data using an interrupt method.*
- void [UART\\_TransferAbortReceive](#) (UART\_Type \*base, uart\_handle\_t \*handle)  
*Aborts the interrupt-driven data receiving.*
- status\_t [UART\\_TransferGetReceiveCount](#) (UART\_Type \*base, uart\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of bytes that have been received.*
- status\_t [UART\\_EnableTxFIFO](#) (UART\_Type \*base, bool enable)  
*Enables or disables the UART Tx FIFO.*
- status\_t [UART\\_EnableRxFIFO](#) (UART\_Type \*base, bool enable)  
*Enables or disables the UART Rx FIFO.*
- static void [UART\\_SetRxFifoWatermark](#) (UART\_Type \*base, uint8\_t water)

- static void [UART\\_SetTxFifoWatermark](#) (UART\_Type \*base, uint8\_t water)
 

*Sets the rx FIFO watermark.*
- void [UART\\_TransferHandleIRQ](#) (UART\_Type \*base, void \*irqHandle)
 

*UART IRQ handle function.*
- void [UART\\_TransferHandleErrorIRQ](#) (UART\_Type \*base, void \*irqHandle)
 

*UART Error IRQ handle function.*

### 33.2.3 Data Structure Documentation

#### 33.2.3.1 struct uart\_config\_t

##### Data Fields

- uint32\_t [baudRate\\_Bps](#)

*UART baud rate.*
- [uart\\_parity\\_mode\\_t parityMode](#)

*Parity mode, disabled (default), even, odd.*
- uint8\_t [txFifoWatermark](#)

*TX FIFO watermark.*
- uint8\_t [rxFifoWatermark](#)

*RX FIFO watermark.*
- bool [enableRxRTS](#)

*RX RTS enable.*
- bool [enableTxCTS](#)

*TX CTS enable.*
- [uart\\_idle\\_type\\_select\\_t idleType](#)

*IDLE type select.*
- bool [enableTx](#)

*Enable TX.*
- bool [enableRx](#)

*Enable RX.*

##### Field Documentation

###### (1) [uart\\_idle\\_type\\_select\\_t uart\\_config\\_t::idleType](#)

#### 33.2.3.2 struct uart\_transfer\_t

##### Data Fields

- size\_t [dataSize](#)

*The byte count to be transfer.*
- uint8\_t \* [data](#)

*The buffer of data to be transfer.*
- uint8\_t \* [rxData](#)

*The buffer to receive data.*
- const uint8\_t \* [txData](#)

*The buffer of data to be sent.*

## Field Documentation

- (1) `uint8_t* uart_transfer_t::data`
- (2) `uint8_t* uart_transfer_t::rxData`
- (3) `const uint8_t* uart_transfer_t::txData`
- (4) `size_t uart_transfer_t::dataSize`

### 33.2.3.3 struct \_uart\_handle

#### Data Fields

- `const uint8_t *volatile txData`  
*Address of remaining data to send.*
- `volatile size_t txDataSize`  
*Size of the remaining data to send.*
- `size_t txDataSizeAll`  
*Size of the data to send out.*
- `uint8_t *volatile rxData`  
*Address of remaining data to receive.*
- `volatile size_t rxDataSize`  
*Size of the remaining data to receive.*
- `size_t rxDataSizeAll`  
*Size of the data to receive.*
- `uint8_t * rxRingBuffer`  
*Start address of the receiver ring buffer.*
- `size_t rxRingBufferSize`  
*Size of the ring buffer.*
- `volatile uint16_t rxRingBufferHead`  
*Index for the driver to store received data into ring buffer.*
- `volatile uint16_t rxRingBufferTail`  
*Index for the user to get data from the ring buffer.*
- `uart_transfer_callback_t callback`  
*Callback function.*
- `void * userData`  
*UART callback function parameter.*
- `volatile uint8_t txState`  
*TX transfer state.*
- `volatile uint8_t rxState`  
*RX transfer state.*

## Field Documentation

- (1) `const uint8_t* volatile uart_handle_t::txData`
- (2) `volatile size_t uart_handle_t::txDataSize`
- (3) `size_t uart_handle_t::txDataSizeAll`

- (4) `uint8_t* volatile uart_handle_t::rxData`
- (5) `volatile size_t uart_handle_t::rxDataSize`
- (6) `size_t uart_handle_t::rxDataSizeAll`
- (7) `uint8_t* uart_handle_t::rxRingBuffer`
- (8) `size_t uart_handle_t::rxRingBufferSize`
- (9) `volatile uint16_t uart_handle_t::rxRingBufferHead`
- (10) `volatile uint16_t uart_handle_t::rxRingBufferTail`
- (11) `uart_transfer_callback_t uart_handle_t::callback`
- (12) `void* uart_handle_t::userData`
- (13) `volatile uint8_t uart_handle_t::txState`

### 33.2.4 Macro Definition Documentation

33.2.4.1 `#define FSL_UART_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))`

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

### 33.2.5 Typedef Documentation

33.2.5.1 `typedef void(* uart_transfer_callback_t)(UART_Type *base, uart_handle_t *handle, status_t status, void *userData)`

### 33.2.6 Enumeration Type Documentation

#### 33.2.6.1 anonymous enum

Enumerator

`kStatus_UART_TxBusy` Transmitter is busy.

`kStatus_UART_RxBusy` Receiver is busy.

`kStatus_UART_TxIdle` UART transmitter is idle.

`kStatus_UART_RxIdle` UART receiver is idle.

`kStatus_UART_TxWatermarkTooLarge` TX FIFO watermark too large.

`kStatus_UART_RxWatermarkTooLarge` RX FIFO watermark too large.

`kStatus_UART_FlagCannotClearManually` UART flag can't be manually cleared.

`kStatus_UART_Error` Error happens on UART.

`kStatus_UART_RxRingBufferOverrun` UART RX software ring buffer overrun.

*kStatus\_UART\_RxHardwareOverrun* UART RX receiver overrun.  
*kStatus\_UART\_NoiseError* UART noise error.  
*kStatus\_UART\_FramingError* UART framing error.  
*kStatus\_UART\_ParityError* UART parity error.  
*kStatus\_UART\_BaudrateNotSupport* Baudrate is not support in current clock source.  
*kStatus\_UART\_IdleLineDetected* UART IDLE line detected.  
*kStatus\_UART\_Timeout* UART times out.

### 33.2.6.2 enum uart\_parity\_mode\_t

Enumerator

*kUART\_ParityDisabled* Parity disabled.  
*kUART\_ParityEven* Parity enabled, type even, bit setting: PE|PT = 10.  
*kUART\_ParityOdd* Parity enabled, type odd, bit setting: PE|PT = 11.

### 33.2.6.3 enum uart\_stop\_bit\_count\_t

Enumerator

*kUART\_OneStopBit* One stop bit.  
*kUART\_TwoStopBit* Two stop bits.

### 33.2.6.4 enum uart\_idle\_type\_select\_t

Enumerator

*kUART\_IdleTypeStartBit* Start counting after a valid start bit.  
*kUART\_IdleTypeStopBit* Start counting after a stop bit.

### 33.2.6.5 enum \_uart\_interrupt\_enable

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

Enumerator

*kUART\_RxActiveEdgeInterruptEnable* RX active edge interrupt.  
*kUART\_TxDataRegEmptyInterruptEnable* Transmit data register empty interrupt.  
*kUART\_TransmissionCompleteInterruptEnable* Transmission complete interrupt.  
*kUART\_RxDataRegFullInterruptEnable* Receiver data register full interrupt.  
*kUART\_IdleLineInterruptEnable* Idle line interrupt.  
*kUART\_RxOverrunInterruptEnable* Receiver overrun interrupt.

*kUART\_NoiseErrorInterruptEnable* Noise error flag interrupt.  
*kUART\_FramingErrorInterruptEnable* Framing error flag interrupt.  
*kUART\_ParityErrorInterruptEnable* Parity error flag interrupt.  
*kUART\_RxFifoOverflowInterruptEnable* RX FIFO overflow interrupt.  
*kUART\_TxFifoOverflowInterruptEnable* TX FIFO overflow interrupt.  
*kUART\_RxFifoUnderflowInterruptEnable* RX FIFO underflow interrupt.

### 33.2.6.6 anonymous enum

This provides constants for the UART status flags for use in the UART functions.

Enumerator

*kUART\_TxDataRegEmptyFlag* TX data register empty flag.  
*kUART\_TransmissionCompleteFlag* Transmission complete flag.  
*kUART\_RxDataRegFullFlag* RX data register full flag.  
*kUART\_IdleLineFlag* Idle line detect flag.  
*kUART\_RxOverrunFlag* RX overrun flag.  
*kUART\_NoiseErrorFlag* RX takes 3 samples of each received bit. If any of these samples differ, noise flag sets  
*kUART\_FramingErrorFlag* Frame error flag, sets if logic 0 was detected where stop bit expected.  
*kUART\_ParityErrorFlag* If parity enabled, sets upon parity error detection.  
*kUART\_RxActiveEdgeFlag* RX pin active edge interrupt flag, sets when active edge detected.  
*kUART\_RxActiveFlag* Receiver Active Flag (RAF), sets at beginning of valid start bit.  
*kUART\_NoiseErrorInRxDataRegFlag* Noisy bit, sets if noise detected.  
*kUART\_ParityErrorInRxDataRegFlag* Parity bit, sets if parity error detected.  
*kUART\_TxFifoEmptyFlag* TXEMPT bit, sets if TX buffer is empty.  
*kUART\_RxFifoEmptyFlag* RXEMPT bit, sets if RX buffer is empty.  
*kUART\_TxFifoOverflowFlag* TXOF bit, sets if TX buffer overflow occurred.  
*kUART\_RxFifoOverflowFlag* RXOF bit, sets if receive buffer overflow.  
*kUART\_RxFifoUnderflowFlag* RXUF bit, sets if receive buffer underflow.

### 33.2.7 Function Documentation

#### 33.2.7.1 `uint32_t UART_GetInstance ( UART_Type * base )`

Parameters

---

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

Returns

UART instance.

### 33.2.7.2 status\_t **UART\_Init** ( **UART\_Type** \* *base*, **const uart\_config\_t** \* *config*, **uint32\_t** *srcClock\_Hz* )

This function configures the UART module with the user-defined settings. The user can configure the configuration structure and also get the default configuration by using the [UART\\_GetDefaultConfig\(\)](#) function. The example below shows how to use this API to configure UART.

```
* uart_config_t uartConfig;
* uartConfig.baudRate_Bps = 115200U;
* uartConfig.parityMode = kUART_ParityDisabled;
* uartConfig.stopBitCount = kUART_OneStopBit;
* uartConfig.txFifoWatermark = 0;
* uartConfig.rxFifoWatermark = 1;
* UART_Init(UART1, &uartConfig, 20000000U);
*
```

Parameters

<i>base</i>	UART peripheral base address.
<i>config</i>	Pointer to the user-defined configuration structure.
<i>srcClock_Hz</i>	UART clock source frequency in HZ.

Return values

<i>kStatus_UART_Baudrate-NotSupport</i>	Baudrate is not support in current clock source.
<i>kStatus_Success</i>	Status UART initialize succeed

### 33.2.7.3 void **UART\_Deinit** ( **UART\_Type** \* *base* )

This function waits for TX complete, disables TX and RX, and disables the UART clock.

Parameters

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

### 33.2.7.4 void UART\_GetDefaultConfig ( *uart\_config\_t* \* *config* )

This function initializes the UART configuration structure to a default value. The default values are as follows. *uartConfig->baudRate\_Bps* = 115200U; *uartConfig->bitCountPerChar* = kUART\_8BitsPerChar; *uartConfig->parityMode* = kUART\_ParityDisabled; *uartConfig->stopBitCount* = kUART\_OneStopBit; *uartConfig->txFifoWatermark* = 0; *uartConfig->rxFifoWatermark* = 1; *uartConfig->idleType* = kUART\_IdleTypeStartBit; *uartConfig->enableTx* = false; *uartConfig->enableRx* = false;

Parameters

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

### 33.2.7.5 status\_t UART\_SetBaudRate ( *UART\_Type* \* *base*, *uint32\_t* *baudRate\_Bps*, *uint32\_t* *srcClock\_Hz* )

This function configures the UART module baud rate. This function is used to update the UART module baud rate after the UART module is initialized by the *UART\_Init*.

```
*   UART\_SetBaudRate(UART1, 115200U, 20000000U);
*
```

Parameters

<i>base</i>	UART peripheral base address.
<i>baudRate_Bps</i>	UART baudrate to be set.
<i>srcClock_Hz</i>	UART clock source frequency in Hz.

Return values

<i>kStatus_UART_Baudrate-NotSupport</i>	Baudrate is not support in the current clock source.
---	--

<i>kStatus_Success</i>	Set baudrate succeeded.
------------------------	-------------------------

### 33.2.7.6 void **UART\_Enable9bitMode** ( **UART\_Type \* base**, **bool enable** )

This function set the 9-bit mode for UART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

<i>base</i>	UART peripheral base address.
<i>enable</i>	true to enable, flase to disable.

### 33.2.7.7 static void **UART\_SetMatchAddress** ( **UART\_Type \* base**, **uint8\_t address1**, **uint8\_t address2** ) [inline], [static]

This function configures the address for UART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receives with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note

Any UART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

<i>base</i>	UART peripheral base address.
<i>address1</i>	UART slave address 1.
<i>address2</i>	UART slave address 2.

### 33.2.7.8 static void **UART\_EnableMatchAddress** ( **UART\_Type \* base**, **bool match1**, **bool match2** ) [inline], [static]

Parameters

<i>base</i>	UART peripheral base address.
<i>match1</i>	true to enable match address1, false to disable.
<i>match2</i>	true to enable match address2, false to disable.

### 33.2.7.9 static void UART\_Set9thTransmitBit ( **UART\_Type** \* *base* ) [inline], [static]

Parameters

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

### 33.2.7.10 static void UART\_Clear9thTransmitBit ( **UART\_Type** \* *base* ) [inline], [static]

Parameters

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

### 33.2.7.11 uint32\_t UART\_GetStatusFlags ( **UART\_Type** \* *base* )

This function gets all UART status flags. The flags are returned as the logical OR value of the enumerators \_uart\_flags. To check a specific status, compare the return value with enumerators in \_uart\_flags. For example, to check whether the TX is empty, do the following.

```
*     if (kUART_TxDataRegEmptyFlag & UART_GetStatusFlags(UART1))
*
*     {
*         ...
*     }
```

Parameters

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

Returns

UART status flags which are ORed by the enumerators in the \_uart\_flags.

### 33.2.7.12 status\_t UART\_ClearStatusFlags ( **UART\_Type** \* *base*, **uint32\_t** *mask* )

This function clears UART status flags with a provided mask. An automatically cleared flag can't be cleared by this function. These flags can only be cleared or set by hardware. kUART\_TxDataRegEmptyFlag, kUART\_TransmissionCompleteFlag, kUART\_RxDataRegFullFlag, kUART\_RxActiveFlag, kUART\_NoiseErrorInRxDataRegFlag, kUART\_ParityErrorInRxDataRegFlag, kUART\_TxFifoEmptyFlag, kUART\_RxFifoEmptyFlag

Note

that this API should be called when the Tx/Rx is idle. Otherwise it has no effect.

Parameters

<i>base</i>	UART peripheral base address.
<i>mask</i>	The status flags to be cleared; it is logical OR value of <code>_uart_flags</code> .

Return values

<i>kStatus_UART_FlagCannotClearManually</i>	The flag can't be cleared by this function but it is cleared automatically by hardware.
<i>kStatus_Success</i>	Status in the mask is cleared.

### 33.2.7.13 void UART\_EnableInterrupts ( **UART\_Type** \* *base*, **uint32\_t** *mask* )

This function enables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [\\_uart\\_interrupt\\_enable](#). For example, to enable TX empty interrupt and RX full interrupt, do the following.

```
*     UART_EnableInterrupts(UART1,
                           kUART_TxDataRegEmptyInterruptEnable |
                           kUART_RxDataRegFullInterruptEnable);
*
```

Parameters

<i>base</i>	UART peripheral base address.
<i>mask</i>	The interrupts to enable. Logical OR of <a href="#">_uart_interrupt_enable</a> .

### 33.2.7.14 void UART\_DisableInterrupts ( **UART\_Type** \* *base*, **uint32\_t** *mask* )

This function disables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [\\_uart\\_interrupt\\_enable](#). For example, to disable TX empty interrupt and RX full interrupt do the following.

```
*     UART_DisableInterrupts(UART1,
*                           kUART_TxDataRegEmptyInterruptEnable |
*                           kUART_RxDataRegFullInterruptEnable);
*
```

## Parameters

<i>base</i>	UART peripheral base address.
<i>mask</i>	The interrupts to disable. Logical OR of <a href="#">_uart_interrupt_enable</a> .

**33.2.7.15 uint32\_t UART\_GetEnabledInterrupts ( **UART\_Type** \* *base* )**

This function gets the enabled UART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators [\\_uart\\_interrupt\\_enable](#). To check a specific interrupts enable status, compare the return value with enumerators in [\\_uart\\_interrupt\\_enable](#). For example, to check whether TX empty interrupt is enabled, do the following.

```
*     uint32_t enabledInterrupts = UART_GetEnabledInterrupts(UART1);
*
*     if (kUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
*     {
*         ...
*     }
```

## Parameters

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

## Returns

UART interrupt flags which are logical OR of the enumerators in [\\_uart\\_interrupt\\_enable](#).

**33.2.7.16 static uint32\_t UART\_GetDataRegisterAddress ( **UART\_Type** \* *base* )  
[**inline**], [**static**]**

This function returns the UART data register address, which is mainly used by DMA/eDMA.

## Parameters

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

Returns

UART data register addresses which are used both by the transmitter and the receiver.

### 33.2.7.17 static void UART\_EnableTxDMA ( **UART\_Type** \* *base*, **bool** *enable* ) [**inline**], [**static**]

This function enables or disables the transmit data register empty flag, S1[TDRE], to generate the DMA requests.

Parameters

<i>base</i>	UART peripheral base address.
<i>enable</i>	True to enable, false to disable.

### 33.2.7.18 static void UART\_EnableRxDMA ( **UART\_Type** \* *base*, **bool** *enable* ) [**inline**], [**static**]

This function enables or disables the receiver data register full flag, S1[RDRF], to generate DMA requests.

Parameters

<i>base</i>	UART peripheral base address.
<i>enable</i>	True to enable, false to disable.

### 33.2.7.19 static void UART\_EnableTx ( **UART\_Type** \* *base*, **bool** *enable* ) [**inline**], [**static**]

This function enables or disables the UART transmitter.

Parameters

<i>base</i>	UART peripheral base address.
<i>enable</i>	True to enable, false to disable.

### 33.2.7.20 static void UART\_EnableRx ( **UART\_Type** \* *base*, **bool** *enable* ) [**inline**], [**static**]

This function enables or disables the UART receiver.

Parameters

<i>base</i>	UART peripheral base address.
<i>enable</i>	True to enable, false to disable.

### 33.2.7.21 static void UART\_WriteByte ( **UART\_Type** \* *base*, **uint8\_t** *data* ) [inline], [static]

This function writes data to the TX register directly. The upper layer must ensure that the TX register is empty or TX FIFO has empty room before calling this function.

Parameters

<i>base</i>	UART peripheral base address.
<i>data</i>	The byte to write.

### 33.2.7.22 static **uint8\_t** UART\_ReadByte ( **UART\_Type** \* *base* ) [inline], [static]

This function reads data from the RX register directly. The upper layer must ensure that the RX register is full or that the TX FIFO has data before calling this function.

Parameters

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

Returns

The byte read from UART data register.

### 33.2.7.23 static **uint8\_t** UART\_GetRxFifoCount ( **UART\_Type** \* *base* ) [inline], [static]

Parameters

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

Returns

rx FIFO data count.

33.2.7.24 **static uint8\_t UART\_GetTxFifoCount( UART\_Type \* *base* ) [inline],  
[static]**

Parameters

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

Returns

tx FIFO data count.

### 33.2.7.25 void UART\_SendAddress ( **UART\_Type** \* *base*, **uint8\_t** *address* )

Parameters

<i>base</i>	UART peripheral base address.
<i>address</i>	UART slave address.

### 33.2.7.26 status\_t UART\_WriteBlocking ( **UART\_Type** \* *base*, **const uint8\_t** \* *data*, **size\_t** *length* )

This function polls the TX register, waits for the TX register to be empty or for the TX FIFO to have room and writes data to the TX buffer.

Parameters

<i>base</i>	UART peripheral base address.
<i>data</i>	Start address of the data to write.
<i>length</i>	Size of the data to write.

Return values

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

### 33.2.7.27 status\_t UART\_ReadBlocking ( **UART\_Type** \* *base*, **uint8\_t** \* *data*, **size\_t** *length* )

This function polls the RX register, waits for the RX register to be full or for RX FIFO to have data, and reads data from the TX register.

Parameters

<i>base</i>	UART peripheral base address.
<i>data</i>	Start address of the buffer to store the received data.
<i>length</i>	Size of the buffer.

Return values

<i>kStatus_UART_Rx-HardwareOverrun</i>	Receiver overrun occurred while receiving data.
<i>kStatus_UART_Noise-Error</i>	A noise error occurred while receiving data.
<i>kStatus_UART_Framing-Error</i>	A framing error occurred while receiving data.
<i>kStatus_UART_Parity-Error</i>	A parity error occurred while receiving data.
<i>kStatus_UART_Timeout</i>	Transmission timed out and was aborted.
<i>kStatus_Success</i>	Successfully received all data.

### 33.2.7.28 void UART\_TransferCreateHandle ( **UART\_Type \* base**, **uart\_handle\_t \* handle**, **uart\_transfer\_callback\_t callback**, **void \* userData** )

This function initializes the UART handle which can be used for other UART transactional APIs. Usually, for a specified UART instance, call this API once to get the initialized handle.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>callback</i>	The callback function.
<i>userData</i>	The parameter of the callback function.

### 33.2.7.29 void UART\_TransferStartRingBuffer ( **UART\_Type \* base**, **uart\_handle\_t \* handle**, **uint8\_t \* ringBuffer**, **size\_t ringBufferSize** )

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received are stored into the ring buffer even when the user doesn't call the [UART\\_TransferReceiveNonBlocking\(\)](#) API. If data is already received in the ring buffer, the user can get the received data from the ring buffer directly.

## Note

When using the RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, only 31 bytes are used for saving data.

## Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>ringBuffer</i>	Start address of the ring buffer for background receiving. Pass NULL to disable the ring buffer.
<i>ringBufferSize</i>	Size of the ring buffer.

### 33.2.7.30 void `UART_TransferStopRingBuffer ( UART_Type * base, uart_handle_t * handle )`

This function aborts the background transfer and uninstalls the ring buffer.

## Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.

### 33.2.7.31 size\_t `UART_TransferGetRxRingBufferLength ( uart_handle_t * handle )`

## Parameters

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

## Returns

Length of received data in RX ring buffer.

### 33.2.7.32 status\_t `UART_TransferSendNonBlocking ( UART_Type * base, uart_handle_t * handle, uart_transfer_t * xfer )`

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in the ISR, the UART driver calls the callback function and passes the `kStatus_UART_TxIdle` as status parameter.

## Note

The kStatus\_UART\_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out. Before disabling the TX, check the kUART\_TransmissionCompleteFlag to ensure that the TX is finished.

## Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>xfer</i>	UART transfer structure. See <a href="#">uart_transfer_t</a> .

## Return values

<i>kStatus_Success</i>	Successfully start the data transmission.
<i>kStatus_UART_TxBusy</i>	Previous transmission still not finished; data not all written to TX register yet.
<i>kStatus_InvalidArgument</i>	Invalid argument.

**33.2.7.33 void UART\_TransferAbortSend ( **UART\_Type** \* *base*, **uart\_handle\_t** \* *handle* )**

This function aborts the interrupt-driven data sending. The user can get the remainBytes to find out how many bytes are not sent out.

## Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.

**33.2.7.34 status\_t UART\_TransferGetSendCount ( **UART\_Type** \* *base*, **uart\_handle\_t** \* *handle*, **uint32\_t** \* *count* )**

This function gets the number of bytes sent out to bus by using the interrupt method.

## Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>count</i>	Send bytes count.

Return values

<i>kStatus_NoTransferInProgress</i>	No send in progress.
<i>kStatus_InvalidArgument</i>	The parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <code>count</code> ;

### 33.2.7.35 `status_t UART_TransferReceiveNonBlocking ( UART_Type * base, uart_handle_t * handle, uart_transfer_t * xfer, size_t * receivedBytes )`

This function receives data using an interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough to read, the receive request is saved by the UART driver. When the new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter [k\\_Status\\_UART\\_RxIdle](#). For example, the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer. The 5 bytes are copied to the `xfer->data` and this function returns with the parameter `receivedBytes` set to 5. For the left 5 bytes, newly arrived data is saved from the `xfer->data[5]`. When 5 bytes are received, the UART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to the `xfer->data`. When all data is received, the upper layer is notified.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>xfer</i>	UART transfer structure, see <a href="#">uart_transfer_t</a> .
<i>receivedBytes</i>	Bytes received from the ring buffer directly.

Return values

<i>kStatus_Success</i>	Successfully queue the transfer into transmit queue.
<i>kStatus_UART_RxBusy</i>	Previous receive request is not finished.
<i>kStatus_InvalidArgument</i>	Invalid argument.

**33.2.7.36 void UART\_TransferAbortReceive ( **UART\_Type** \* *base*, **uart\_handle\_t** \* *handle* )**

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to know how many bytes are not received yet.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.

### 33.2.7.37 status\_t UART\_TransferGetReceiveCount ( **UART\_Type** \* *base*, **uart\_handle\_t** \* *handle*, **uint32\_t** \* *count* )

This function gets the number of bytes that have been received.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>count</i>	Receive bytes count.

Return values

<i>kStatus_NoTransferInProgress</i>	No receive in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <i>count</i> ;

### 33.2.7.38 status\_t UART\_EnableTxFIFO ( **UART\_Type** \* *base*, **bool** *enable* )

This function enables or disables the UART Tx FIFO.

param *base* UART peripheral base address. param *enable* true to enable, false to disable. retval *kStatus\_Success* Successfully turn on or turn off Tx FIFO. retval *kStatus\_Fail* Fail to turn on or turn off Tx FIFO.

### 33.2.7.39 status\_t UART\_EnableRxFIFO ( **UART\_Type** \* *base*, **bool** *enable* )

This function enables or disables the UART Rx FIFO.

param *base* UART peripheral base address. param *enable* true to enable, false to disable. retval *kStatus\_Success* Successfully turn on or turn off Rx FIFO. retval *kStatus\_Fail* Fail to turn on or turn off Rx FIFO.

### 33.2.7.40 static void UART\_SetRxFifoWatermark ( **UART\_Type** \* *base*, **uint8\_t** *water* ) [inline], [static]

Parameters

<i>base</i>	UART peripheral base address.
<i>water</i>	Rx FIFO watermark.

### 33.2.7.41 static void UART\_SetTxFifoWatermark ( **UART\_Type** \* *base*, **uint8\_t** *water* ) [**inline**], [**static**]

Parameters

<i>base</i>	UART peripheral base address.
<i>water</i>	Tx FIFO watermark.

### 33.2.7.42 void UART\_TransferHandleIRQ ( **UART\_Type** \* *base*, **void** \* *irqHandle* )

This function handles the UART transmit and receive IRQ request.

Parameters

<i>base</i>	UART peripheral base address.
<i>irqHandle</i>	UART handle pointer.

### 33.2.7.43 void UART\_TransferHandleErrorIRQ ( **UART\_Type** \* *base*, **void** \* *irqHandle* )

This function handles the UART error IRQ request.

Parameters

<i>base</i>	UART peripheral base address.
<i>irqHandle</i>	UART handle pointer.

## 33.2.8 Variable Documentation

### 33.2.8.1 **void\*** *s\_uartHandle*[]

### 33.2.8.2 **uart\_isr\_t** *s\_uartIsr*

## 33.3 UART DMA Driver

### 33.3.1 Overview

#### Data Structures

- struct [uart\\_dma\\_handle\\_t](#)  
*UART DMA handle. [More...](#)*

#### Typedefs

- [typedef void\(\\* uart\\_dma\\_transfer\\_callback\\_t \)\(UART\\_Type \\*base, uart\\_dma\\_handle\\_t \\*handle, status\\_t status, void \\*userData\)](#)  
*UART transfer callback function.*

#### Driver version

- [#define FSL\\_UART\\_DMA\\_DRIVER\\_VERSION \(MAKE\\_VERSION\(2, 5, 0\)\)](#)  
*UART DMA driver version.*

#### eDMA transactional

- [void UART\\_TransferCreateHandleDMA \(UART\\_Type \\*base, uart\\_dma\\_handle\\_t \\*handle, uart\\_dma\\_transfer\\_callback\\_t callback, void \\*userData, dma\\_handle\\_t \\*txDmaHandle, dma\\_handle\\_t \\*rxDmaHandle\)](#)  
*Initializes the UART handle which is used in transactional functions and sets the callback.*
- [status\\_t UART\\_TransferSendDMA \(UART\\_Type \\*base, uart\\_dma\\_handle\\_t \\*handle, uart\\_transfer\\_t \\*xfer\)](#)  
*Sends data using DMA.*
- [status\\_t UART\\_TransferReceiveDMA \(UART\\_Type \\*base, uart\\_dma\\_handle\\_t \\*handle, uart\\_transfer\\_t \\*xfer\)](#)  
*Receives data using DMA.*
- [void UART\\_TransferAbortSendDMA \(UART\\_Type \\*base, uart\\_dma\\_handle\\_t \\*handle\)](#)  
*Aborts the send data using DMA.*
- [void UART\\_TransferAbortReceiveDMA \(UART\\_Type \\*base, uart\\_dma\\_handle\\_t \\*handle\)](#)  
*Aborts the received data using DMA.*
- [status\\_t UART\\_TransferGetSendCountDMA \(UART\\_Type \\*base, uart\\_dma\\_handle\\_t \\*handle, uint32\\_t \\*count\)](#)  
*Gets the number of bytes written to UART TX register.*
- [status\\_t UART\\_TransferGetReceiveCountDMA \(UART\\_Type \\*base, uart\\_dma\\_handle\\_t \\*handle, uint32\\_t \\*count\)](#)  
*Gets the number of bytes that have been received.*
- [void UART\\_TransferDMAHandleIRQ \(UART\\_Type \\*base, void \\*uartDmaHandle\)](#)  
*UART DMA IRQ handle function.*

### 33.3.2 Data Structure Documentation

#### 33.3.2.1 struct \_uart\_dma\_handle

##### Data Fields

- `UART_Type *base`  
*UART peripheral base address.*
- `uart_dma_transfer_callback_t callback`  
*Callback function.*
- `void *userData`  
*UART callback function parameter.*
- `size_t rxDataSizeAll`  
*Size of the data to receive.*
- `size_t txDataSizeAll`  
*Size of the data to send out.*
- `dma_handle_t *txDmaHandle`  
*The DMA TX channel used.*
- `dma_handle_t *rxDmaHandle`  
*The DMA RX channel used.*
- `volatile uint8_t txState`  
*TX transfer state.*
- `volatile uint8_t rxState`  
*RX transfer state.*

##### Field Documentation

- (1) `UART_Type* uart_dma_handle_t::base`
- (2) `uart_dma_transfer_callback_t uart_dma_handle_t::callback`
- (3) `void* uart_dma_handle_t::userData`
- (4) `size_t uart_dma_handle_t::rxDataSizeAll`
- (5) `size_t uart_dma_handle_t::txDataSizeAll`
- (6) `dma_handle_t* uart_dma_handle_t::txDmaHandle`
- (7) `dma_handle_t* uart_dma_handle_t::rxDmaHandle`
- (8) `volatile uint8_t uart_dma_handle_t::txState`

### 33.3.3 Macro Definition Documentation

#### 33.3.3.1 #define FSL\_UART\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 0))

### 33.3.4 Typedef Documentation

**33.3.4.1 `typedef void(* uart_dma_transfer_callback_t)(UART_Type *base, uart_dma_handle_t *handle, status_t status, void *userData)`**

### 33.3.5 Function Documentation

**33.3.5.1 `void UART_TransferCreateHandleDMA ( UART_Type * base, uart_dma_handle_t * handle, uart_dma_transfer_callback_t callback, void * userData, dma_handle_t * txDmaHandle, dma_handle_t * rxDmaHandle )`**

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	Pointer to the <code>uart_dma_handle_t</code> structure.
<i>callback</i>	UART callback, NULL means no callback.
<i>userData</i>	User callback function data.
<i>rxDmaHandle</i>	User requested DMA handle for the RX DMA transfer.
<i>txDmaHandle</i>	User requested DMA handle for the TX DMA transfer.

**33.3.5.2 `status_t UART_TransferSendDMA ( UART_Type * base, uart_dma_handle_t * handle, uart_transfer_t * xfer )`**

This function sends data using DMA. This is non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>xfer</i>	UART DMA transfer structure. See <a href="#">uart_transfer_t</a> .

Return values

<i>kStatus_Success</i>	if succeeded; otherwise failed.
<i>kStatus_UART_TxBusy</i>	Previous transfer ongoing.
<i>kStatus_InvalidArgument</i>	Invalid argument.

### 33.3.5.3 status\_t **UART\_TransferReceiveDMA** ( **UART\_Type \* base**, **uart\_dma\_handle\_t \* handle**, **uart\_transfer\_t \* xfer** )

This function receives data using DMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	Pointer to the <code>uart_dma_handle_t</code> structure.
<i>xfer</i>	UART DMA transfer structure. See <a href="#">uart_transfer_t</a> .

Return values

<i>kStatus_Success</i>	if succeeded; otherwise failed.
<i>kStatus_UART_RxBusy</i>	Previous transfer on going.
<i>kStatus_InvalidArgument</i>	Invalid argument.

### 33.3.5.4 void `UART_TransferAbortSendDMA` ( `UART_Type * base, uart_dma_handle_t * handle` )

This function aborts the sent data using DMA.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	Pointer to <code>uart_dma_handle_t</code> structure.

### 33.3.5.5 void `UART_TransferAbortReceiveDMA` ( `UART_Type * base, uart_dma_handle_t * handle` )

This function abort receive data which using DMA.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	Pointer to <code>uart_dma_handle_t</code> structure.

### 33.3.5.6 status\_t `UART_TransferGetSendCountDMA` ( `UART_Type * base, uart_dma_handle_t * handle, uint32_t * count` )

This function gets the number of bytes written to UART TX register by DMA.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>count</i>	Send bytes count.

Return values

<i>kStatus_NoTransferIn-Progress</i>	No send in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <i>count</i> ;

### 33.3.5.7 **status\_t UART\_TransferGetReceiveCountDMA ( *UART\_Type \* base, uart\_dma\_handle\_t \* handle, uint32\_t \* count* )**

This function gets the number of bytes that have been received.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>count</i>	Receive bytes count.

Return values

<i>kStatus_NoTransferIn-Progress</i>	No receive in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <i>count</i> ;

### 33.3.5.8 **void UART\_TransferDMAHandleIRQ ( *UART\_Type \* base, void \* uartDmaHandle* )**

This function handles the UART transmit complete IRQ request and invoke user callback.

## Parameters

<i>base</i>	UART peripheral base address.
<i>uartDma-Handle</i>	UART handle pointer.

## 33.4 UART FreeRTOS Driver

### 33.4.1 Overview

#### Data Structures

- struct `uart_rtos_config_t`  
*UART configuration structure.* [More...](#)

#### Driver version

- #define `FSL_UART_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 5, 0)`)  
*UART FreeRTOS driver version.*

#### UART RTOS Operation

- int `UART_RTOS_Init` (`uart_rtos_handle_t *handle, uart_handle_t *t_handle, const uart_rtos_config_t *cfg`)  
*Initializes a UART instance for operation in RTOS.*
- int `UART_RTOS_Deinit` (`uart_rtos_handle_t *handle`)  
*Deinitializes a UART instance for operation.*

#### UART transactional Operation

- int `UART_RTOS_Send` (`uart_rtos_handle_t *handle, uint8_t *buffer, uint32_t length`)  
*Sends data in the background.*
- int `UART_RTOS_Receive` (`uart_rtos_handle_t *handle, uint8_t *buffer, uint32_t length, size_t *received`)  
*Receives data.*

### 33.4.2 Data Structure Documentation

#### 33.4.2.1 struct `uart_rtos_config_t`

##### Data Fields

- `UART_Type * base`  
*UART base address.*
- `uint32_t srclk`  
*UART source clock in Hz.*
- `uint32_t baudrate`  
*Desired communication speed.*
- `uart_parity_mode_t parity`  
*Parity setting.*

- `uart_stop_bit_count_t stopbits`  
*Number of stop bits to use.*
- `uint8_t * buffer`  
*Buffer for background reception.*
- `uint32_t buffer_size`  
*Size of buffer for background reception.*

### 33.4.3 Macro Definition Documentation

#### 33.4.3.1 `#define FSL_UART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))`

### 33.4.4 Function Documentation

#### 33.4.4.1 `int UART_RTOS_Init ( uart_rtos_handle_t * handle, uart_handle_t * t_handle, const uart_rtos_config_t * cfg )`

Parameters

<code>handle</code>	The RTOS UART handle, the pointer to an allocated space for RTOS context.
<code>t_handle</code>	The pointer to the allocated space to store the transactional layer internal state.
<code>cfg</code>	The pointer to the parameters required to configure the UART after initialization.

Returns

0 succeed; otherwise fail.

#### 33.4.4.2 `int UART_RTOS_Deinit ( uart_rtos_handle_t * handle )`

This function deinitializes the UART module, sets all register values to reset value, and frees the resources.

Parameters

<code>handle</code>	The RTOS UART handle.
---------------------	-----------------------

#### 33.4.4.3 `int UART_RTOS_Send ( uart_rtos_handle_t * handle, uint8_t * buffer, uint32_t length )`

This function sends data. It is a synchronous API. If the hardware buffer is full, the task is in the blocked state.

Parameters

<i>handle</i>	The RTOS UART handle.
<i>buffer</i>	The pointer to the buffer to send.
<i>length</i>	The number of bytes to send.

#### 33.4.4.4 int UART\_RTOS\_Receive ( *uart\_rtos\_handle\_t \* handle*, *uint8\_t \* buffer*, *uint32\_t length*, *size\_t \* received* )

This function receives data from UART. It is a synchronous API. If data is immediately available, it is returned immediately and the number of bytes received.

Parameters

<i>handle</i>	The RTOS UART handle.
<i>buffer</i>	The pointer to the buffer to write received data.
<i>length</i>	The number of bytes to receive.
<i>received</i>	The pointer to a variable of size_t where the number of received data is filled.

## 33.5 UART CMSIS Driver

This section describes the programming interface of the UART Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method see <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

The UART driver includes transactional APIs.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements please write custom code.

### 33.5.1 UART CMSIS Driver

#### 33.5.1.1 UART Send/receive using an interrupt method

```
/* UART callback */
void UART_Callback(uint32_t event)
{
    if (event == ARM_USART_EVENT_SEND_COMPLETE)
    {
        txBufferFull = false;
        txOnGoing = false;
    }

    if (event == ARM_USART_EVENT_RECEIVE_COMPLETE)
    {
        rxBufferEmpty = false;
        rxOnGoing = false;
    }
}
Driver_USART0.Initialize(UART_Callback);
Driver_USART0.PowerControl(ARM_POWER_FULL);
/* Send g_tipString out. */
txOnGoing = true;
Driver_USART0.Send(g_tipString, sizeof(g_tipString) - 1);

/* Wait send finished */
while (txOnGoing)
{
}
```

#### 33.5.1.2 UART Send/Receive using the DMA method

```
/* UART callback */
void UART_Callback(uint32_t event)
{
    if (event == ARM_USART_EVENT_SEND_COMPLETE)
    {
        txBufferFull = false;
        txOnGoing = false;
    }

    if (event == ARM_USART_EVENT_RECEIVE_COMPLETE)
```

```
{  
    rxBufferEmpty = false;  
    rxOnGoing = false;  
}  
}  
  
Driver_USART0.Initialize(UART_Callback);  
DMAMGR_Init();  
Driver_USART0.PowerControl(ARM_POWER_FULL);  
  
/* Send g_tipString out. */  
txOnGoing = true;  
  
Driver_USART0.Send(g_tipString, sizeof(g_tipString) - 1);  
  
/* Wait send finished */  
while (txOnGoing)  
{  
}
```

# Chapter 34

## VREF: Voltage Reference Driver

### 34.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Crossbar Voltage Reference (VREF) block of MCUXpresso SDK devices.

The Voltage Reference(VREF) supplies an accurate 1.2 V voltage output that can be trimmed in 0.5 mV steps. VREF can be used in applications to provide a reference voltage to external devices and to internal analog peripherals, such as the ADC, DAC, or CMP. The voltage reference has operating modes that provide different levels of supply rejection and power consumption.

### 34.2 VREF functional Operation

To configure the VREF driver, configure `vref_config_t` structure in one of two ways.

1. Use the `VREF_GetDefaultConfig()` function.
2. Set the parameter in the `vref_config_t` structure.

To initialize the VREF driver, call the `VREF_Init()` function and pass a pointer to the `vref_config_t` structure.

To de-initialize the VREF driver, call the `VREF_Deinit()` function.

### 34.3 Typical use case and example

This example shows how to generate a reference voltage by using the VREF module.

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

## Data Structures

- struct `vref_config_t`  
*The description structure for the VREF module. [More...](#)*

## Enumerations

- enum `vref_buffer_mode_t` {  
    `kVREF_ModeBandgapOnly` = 0U,  
    `kVREF_ModeHighPowerBuffer` = 1U,  
    `kVREF_ModeLowPowerBuffer` = 2U }  
*VREF modes.*

## Driver version

- #define `FSL_VREF_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 2)`)  
*Version 2.1.2.*

## VREF functional operation

- void [VREF\\_Init](#) (VREF\_Type \*base, const vref\_config\_t \*config)  
*Enables the clock gate and configures the VREF module according to the configuration structure.*
- void [VREF\\_Deinit](#) (VREF\_Type \*base)  
*Stops and disables the clock for the VREF module.*
- void [VREF\\_GetDefaultConfig](#) (vref\_config\_t \*config)  
*Initializes the VREF configuration structure.*
- void [VREF\\_SetTrimVal](#) (VREF\_Type \*base, uint8\_t trimValue)  
*Sets a TRIM value for the reference voltage.*
- static uint8\_t [VREF\\_GetTrimVal](#) (VREF\_Type \*base)  
*Reads the value of the TRIM meaning output voltage.*
- void [VREF\\_SetLowReferenceTrimVal](#) (VREF\_Type \*base, uint8\_t trimValue)  
*Sets the TRIM value for the low voltage reference.*
- static uint8\_t [VREF\\_GetLowReferenceTrimVal](#) (VREF\_Type \*base)  
*Reads the value of the TRIM meaning output voltage.*

## 34.4 Data Structure Documentation

### 34.4.1 struct vref\_config\_t

#### Data Fields

- [vref\\_buffer\\_mode\\_t bufferMode](#)  
*Buffer mode selection.*
- bool [enableLowRef](#)  
*Set VREFL (0.4 V) reference buffer enable or disable.*
- bool [enableExternalVoltRef](#)  
*Select external voltage reference or not (internal)*

## 34.5 Macro Definition Documentation

### 34.5.1 #define FSL\_VREF\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 2))

## 34.6 Enumeration Type Documentation

### 34.6.1 enum vref\_buffer\_mode\_t

Enumerator

*kVREF\_ModeBandgapOnly* Bandgap on only, for stabilization and startup.

*kVREF\_ModeHighPowerBuffer* High-power buffer mode enabled.

*kVREF\_ModeLowPowerBuffer* Low-power buffer mode enabled.

## 34.7 Function Documentation

### 34.7.1 void VREF\_Init ( VREF\_Type \* *base*, const vref\_config\_t \* *config* )

This function must be called before calling all other VREF driver functions, read/write registers, and configurations with user-defined settings. The example below shows how to set up [vref\\_config\\_t](#).

**t** parameters and how to call the VREF\_Init function by passing in these parameters. This is an example.

```
*   vref_config_t vrefConfig;
*   vrefConfig.bufferMode = kVREF_ModeHighPowerBuffer;
*   vrefConfig.enableExternalVoltRef = false;
*   vrefConfig.enableLowRef = false;
*   VREF_Init(VREF, &vrefConfig);
*
```

Parameters

<i>base</i>	VREF peripheral address.
<i>config</i>	Pointer to the configuration structure.

### 34.7.2 void VREF\_Deinit ( VREF\_Type \* *base* )

This function should be called to shut down the module. This is an example.

```
*   vref_config_t vrefUserConfig;
*   VREF_Init(VREF);
*   VREF_GetDefaultConfig(&vrefUserConfig);
*   ...
*   VREF_Deinit(VREF);
*
```

Parameters

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

### 34.7.3 void VREF\_GetDefaultConfig ( vref\_config\_t \* *config* )

This function initializes the VREF configuration structure to default values. This is an example.

```
*   vrefConfig->bufferMode = kVREF_ModeHighPowerBuffer;
*   vrefConfig->enableExternalVoltRef = false;
*   vrefConfig->enableLowRef = false;
*
```

Parameters

<i>config</i>	Pointer to the initialization structure.
---------------	--

#### 34.7.4 void VREF\_SetTrimVal ( VREF\_Type \* *base*, uint8\_t *trimValue* )

This function sets a TRIM value for the reference voltage. Note that the TRIM value maximum is 0x3F.

Parameters

<i>base</i>	VREF peripheral address.
<i>trimValue</i>	Value of the trim register to set the output reference voltage (maximum 0x3F (6-bit)).

#### 34.7.5 static uint8\_t VREF\_GetTrimVal ( VREF\_Type \* *base* ) [inline], [static]

This function gets the TRIM value from the TRM register.

Parameters

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

Returns

Six-bit value of trim setting.

#### 34.7.6 void VREF\_SetLowReferenceTrimVal ( VREF\_Type \* *base*, uint8\_t *trimValue* )

This function sets the TRIM value for low reference voltage. Note the following.

- The TRIM value maximum is 0x05U
- The values 111b and 110b are not valid/allowed.

Parameters

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

<i>trimValue</i>	Value of the trim register to set output low reference voltage (maximum 0x05U (3-bit)).
------------------	---

### 34.7.7 static uint8\_t VREF\_GetLowReferenceTrimVal ( VREF\_Type \* *base* ) [inline], [static]

This function gets the TRIM value from the VREFL\_TRM register.

Parameters

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

Returns

Three-bit value of the trim setting.

# Chapter 35

## WDOG: Watchdog Timer Driver

### 35.1 Overview

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

### 35.2 Typical use case

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

### Data Structures

- struct `wdog_work_mode_t`  
*Defines WDOG work mode. [More...](#)*
- struct `wdog_config_t`  
*Describes WDOG configuration structure. [More...](#)*
- struct `wdog_test_config_t`  
*Describes WDOG test mode configuration structure. [More...](#)*

### Enumerations

- enum `wdog_clock_source_t`{  
  `kWDOG_LpoClockSource` = 0U,  
  `kWDOG_AlternateClockSource` = 1U }  
*Describes WDOG clock source.*
- enum `wdog_clock_prescaler_t`{  
  `kWDOG_ClockPrescalerDivide1` = 0x0U,  
  `kWDOG_ClockPrescalerDivide2` = 0x1U,  
  `kWDOG_ClockPrescalerDivide3` = 0x2U,  
  `kWDOG_ClockPrescalerDivide4` = 0x3U,  
  `kWDOG_ClockPrescalerDivide5` = 0x4U,  
  `kWDOG_ClockPrescalerDivide6` = 0x5U,  
  `kWDOG_ClockPrescalerDivide7` = 0x6U,  
  `kWDOG_ClockPrescalerDivide8` = 0x7U }  
*Describes the selection of the clock prescaler.*
- enum `wdog_test_mode_t`{  
  `kWDOG_QuickTest` = 0U,  
  `kWDOG_ByeTest` = 1U }  
*Describes WDOG test mode.*
- enum `wdog_tested_byte_t`{  
  `kWDOG_TestByte0` = 0U,  
  `kWDOG_TestByte1` = 1U,  
  `kWDOG_TestByte2` = 2U,

```
kWDOG_TestByte3 = 3U }
```

*Describes WDOG tested byte selection in byte test mode.*

- enum `_wdog_interrupt_enable_t` { `kWDOG_InterruptEnable` = WDOG\_STCTRLH\_IRQRSTEN\_-  
MASK }
- WDOG interrupt configuration structure, default settings all disabled.*
- enum `_wdog_status_flags_t` {  
    `kWDOG_RunningFlag` = WDOG\_STCTRLH\_WDOGEN\_MASK,  
    `kWDOG_TimeoutFlag` = WDOG\_STCTRLL\_INTFLG\_MASK }
- WDOG status flags.*

## Driver version

- #define `FSL_WDOG_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 1)`)  
*Defines WDOG driver version 2.0.1.*

## Unlock sequence

- #define `WDOG_FIRST_WORD_OF_UNLOCK` (0xC520U)  
*First word of unlock sequence.*
- #define `WDOG_SECOND_WORD_OF_UNLOCK` (0xD928U)  
*Second word of unlock sequence.*

## Refresh sequence

- #define `WDOG_FIRST_WORD_OF_REFRESH` (0xA602U)  
*First word of refresh sequence.*
- #define `WDOG_SECOND_WORD_OF_REFRESH` (0xB480U)  
*Second word of refresh sequence.*

## WDOG Initialization and De-initialization

- void `WDOG_GetDefaultConfig` (`wdog_config_t` \*config)  
*Initializes the WDOG configuration structure.*
- void `WDOG_Init` (`WDOG_Type` \*base, const `wdog_config_t` \*config)  
*Initializes the WDOG.*
- void `WDOG_Deinit` (`WDOG_Type` \*base)  
*Shuts down the WDOG.*
- void `WDOG_SetTestModeConfig` (`WDOG_Type` \*base, `wdog_test_config_t` \*config)  
*Configures the WDOG functional test.*

## WDOG Functional Operation

- static void `WDOG_Enable` (`WDOG_Type` \*base)  
*Enables the WDOG module.*
- static void `WDOG_Disable` (`WDOG_Type` \*base)  
*Disables the WDOG module.*
- static void `WDOG_EnableInterrupts` (`WDOG_Type` \*base, `uint32_t` mask)  
*Enables the WDOG interrupt.*
- static void `WDOG_DisableInterrupts` (`WDOG_Type` \*base, `uint32_t` mask)  
*Disables the WDOG interrupt.*

- `uint32_t WDOG_GetStatusFlags (WDOG_Type *base)`  
*Gets the WDOG all status flags.*
- `void WDOG_ClearStatusFlags (WDOG_Type *base, uint32_t mask)`  
*Clears the WDOG flag.*
- `static void WDOG_SetTimeoutValue (WDOG_Type *base, uint32_t timeoutCount)`  
*Sets the WDOG timeout value.*
- `static void WDOG_SetWindowValue (WDOG_Type *base, uint32_t windowValue)`  
*Sets the WDOG window value.*
- `static void WDOG_Unlock (WDOG_Type *base)`  
*Unlocks the WDOG register written.*
- `void WDOG_Refresh (WDOG_Type *base)`  
*Refreshes the WDOG timer.*
- `static uint16_t WDOG_GetResetCount (WDOG_Type *base)`  
*Gets the WDOG reset count.*
- `static void WDOG_ClearResetCount (WDOG_Type *base)`  
*Clears the WDOG reset count.*

### 35.3 Data Structure Documentation

#### 35.3.1 struct wdog\_work\_mode\_t

##### Data Fields

- `bool enableStop`  
*Enables or disables WDOG in stop mode.*
- `bool enableDebug`  
*Enables or disables WDOG in debug mode.*

#### 35.3.2 struct wdog\_config\_t

##### Data Fields

- `bool enableWdog`  
*Enables or disables WDOG.*
- `wdog_clock_source_t clockSource`  
*Clock source select.*
- `wdog_clock_prescaler_t prescaler`  
*Clock prescaler value.*
- `wdog_work_mode_t workMode`  
*Configures WDOG work mode in debug stop and wait mode.*
- `bool enableUpdate`  
*Update write-once register enable.*
- `bool enableInterrupt`  
*Enables or disables WDOG interrupt.*
- `bool enableWindowMode`  
*Enables or disables WDOG window mode.*
- `uint32_t windowValue`  
*Window value.*

- `uint32_t timeoutValue`  
*Timeout value.*

### 35.3.3 struct wdog\_test\_config\_t

#### Data Fields

- `wdog_test_mode_t testMode`  
*Selects test mode.*
- `wdog_tested_byte_t testedByte`  
*Selects tested byte in byte test mode.*
- `uint32_t timeoutValue`  
*Timeout value.*

## 35.4 Macro Definition Documentation

### 35.4.1 #define FSL\_WDOG\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

## 35.5 Enumeration Type Documentation

### 35.5.1 enum wdog\_clock\_source\_t

Enumerator

`kWDOG_LpoClockSource` WDOG clock sourced from LPO.

`kWDOG_AlternateClockSource` WDOG clock sourced from alternate clock source.

### 35.5.2 enum wdog\_clock\_prescaler\_t

Enumerator

- |  |               |
|--|---------------|
| <code>kWDOG_ClockPrescalerDivide1</code> | Divided by 1. |
| <code>kWDOG_ClockPrescalerDivide2</code> | Divided by 2. |
| <code>kWDOG_ClockPrescalerDivide3</code> | Divided by 3. |
| <code>kWDOG_ClockPrescalerDivide4</code> | Divided by 4. |
| <code>kWDOG_ClockPrescalerDivide5</code> | Divided by 5. |
| <code>kWDOG_ClockPrescalerDivide6</code> | Divided by 6. |
| <code>kWDOG_ClockPrescalerDivide7</code> | Divided by 7. |
| <code>kWDOG_ClockPrescalerDivide8</code> | Divided by 8. |

### 35.5.3 enum wdog\_test\_mode\_t

Enumerator

*kWDOG\_QuickTest* Selects quick test.

*kWDOG\_Bytetest* Selects byte test.

### 35.5.4 enum wdog\_tested\_byte\_t

Enumerator

*kWDOG\_TestByte0* Byte 0 selected in byte test mode.

*kWDOG\_TestByte1* Byte 1 selected in byte test mode.

*kWDOG\_TestByte2* Byte 2 selected in byte test mode.

*kWDOG\_TestByte3* Byte 3 selected in byte test mode.

### 35.5.5 enum \_wdog\_interrupt\_enable\_t

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

Enumerator

*kWDOG\_InterruptEnable* WDOG timeout generates an interrupt before reset.

### 35.5.6 enum \_wdog\_status\_flags\_t

This structure contains the WDOG status flags for use in the WDOG functions.

Enumerator

*kWDOG\_RunningFlag* Running flag, set when WDOG is enabled.

*kWDOG\_TimeoutFlag* Interrupt flag, set when an exception occurs.

## 35.6 Function Documentation

### 35.6.1 void WDOG\_GetDefaultConfig ( wdog\_config\_t \* config )

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

```

*   wdogConfig->enableWdog = true;
*   wdogConfig->clockSource = kWDOG_IpoClockSource;
*   wdogConfig->prescaler = kWDOG_ClockPrescalerDivide1;
*   wdogConfig->workMode.enableWait = true;
*   wdogConfig->workMode.enableStop = false;
*   wdogConfig->workMode.enableDebug = false;
*   wdogConfig->enableUpdate = true;
*   wdogConfig->enableInterrupt = false;
*   wdogConfig->enableWindowMode = false;
*   wdogConfig->windowValue = 0;
*   wdogConfig->timeoutValue = 0xFFFFU;
*

```

## Parameters

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

## See Also

[wdog\\_config\\_t](#)

**35.6.2 void WDOG\_Init ( WDOG\_Type \* *base*, const wdog\_config\_t \* *config* )**

This function initializes the WDOG. When called, the WDOG runs according to the configuration. To reconfigure WDOG without forcing a reset first, enableUpdate must be set to true in the configuration.

This is an example.

```

*   wdog_config_t config;
*   WDOG_GetDefaultConfig(&config);
*   config.timeoutValue = 0x7ffU;
*   config.enableUpdate = true;
*   WDOG_Init(wdog_base,&config);
*

```

## Parameters

<i>base</i>	WDOG peripheral base address
<i>config</i>	The configuration of WDOG

**35.6.3 void WDOG\_Deinit ( WDOG\_Type \* *base* )**

This function shuts down the WDOG. Ensure that the WDOG\_STCTRLH.ALLOWUPDATE is 1 which indicates that the register update is enabled.

### 35.6.4 void WDOG\_SetTestModeConfig ( WDOG\_Type \* *base*, wdog\_test\_config\_t \* *config* )

This function is used to configure the WDOG functional test. When called, the WDOG goes into test mode and runs according to the configuration. Ensure that the WDOG\_STCTRLH.ALLOWUPDATE is 1 which means that the register update is enabled.

This is an example.

```
*     wdog_test_config_t test_config;
*     test_config.testMode = kWDOG_QuickTest;
*     test_config.timeoutValue = 0xfffffu;
*     WDOG_SetTestModeConfig(wdog_base, &test_config);
*
```

Parameters

<i>base</i>	WDOG peripheral base address
<i>config</i>	The functional test configuration of WDOG

### 35.6.5 static void WDOG\_Enable ( WDOG\_Type \* *base* ) [inline], [static]

This function write value into WDOG\_STCTRLH register to enable the WDOG, it is a write-once register, make sure that the WCT window is still open and this register has not been written in this WCT while this function is called.

Parameters

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

### 35.6.6 static void WDOG\_Disable ( WDOG\_Type \* *base* ) [inline], [static]

This function writes a value into the WDOG\_STCTRLH register to disable the WDOG. It is a write-once register. Ensure that the WCT window is still open and that register has not been written to in this WCT while the function is called.

Parameters

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

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

This function writes a value into the WDOG\_STCTRLH register to enable the WDOG interrupt. It is a write-once register. Ensure that the WCT window is still open and the register has not been written to in this WCT while the function is called.

Parameters

<i>base</i>	WDOG peripheral base address
<i>mask</i>	The interrupts to enable The parameter can be combination of the following source if defined. <ul style="list-style-type: none"> <li>• kWDOG_InterruptEnable</li> </ul>

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

This function writes a value into the WDOG\_STCTRLH register to disable the WDOG interrupt. It is a write-once register. Ensure that the WCT window is still open and the register has not been written to in this WCT while the function is called.

Parameters

<i>base</i>	WDOG peripheral base address
<i>mask</i>	The interrupts to disable The parameter can be combination of the following source if defined. <ul style="list-style-type: none"> <li>• kWDOG_InterruptEnable</li> </ul>

### 35.6.9 **uint32\_t** WDOG\_GetStatusFlags ( **WDOG\_Type** \* *base* )

This function gets all status flags.

This is an example for getting the Running Flag.

```
*     uint32_t status;
*     status = WDOG_GetStatusFlags (wdog_base) &
*                           kWDOG_RunningFlag;
```

\*

## Parameters

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

## Returns

State of the status flag: asserted (true) or not-asserted (false).

## See Also

[\\_wdog\\_status\\_flags\\_t](#)

- true: a related status flag has been set.
- false: a related status flag is not set.

**35.6.10 void WDOG\_ClearStatusFlags ( WDOG\_Type \* *base*, uint32\_t *mask* )**

This function clears the WDOG status flag.

This is an example for clearing the timeout (interrupt) flag.

```
*   WDOG_ClearStatusFlags (wdog_base, kWDOG_TimeoutFlag);
*
```

## Parameters

<i>base</i>	WDOG peripheral base address
<i>mask</i>	The status flags to clear. The parameter could be any combination of the following values. kWDOG_TimeoutFlag

**35.6.11 static void WDOG\_SetTimeoutValue ( WDOG\_Type \* *base*, uint32\_t *timeoutCount* ) [inline], [static]**

This function sets the timeout value. It should be ensured that the time-out value for the WDOG is always greater than 2xWCT time + 20 bus clock cycles. This function writes a value into WDOG\_TOVALH and WDOG\_TOVALL registers which are write-once. Ensure the WCT window is still open and the two registers have not been written to in this WCT while the function is called.

Parameters

<i>base</i>	WDOG peripheral base address
<i>timeoutCount</i>	WDOG timeout value; count of WDOG clock tick.

### 35.6.12 static void WDOG\_SetWindowValue ( WDOG\_Type \* *base*, uint32\_t *windowValue* ) [inline], [static]

This function sets the WDOG window value. This function writes a value into WDOG\_WINH and WDOG\_WINL registers which are write-once. Ensure the WCT window is still open and the two registers have not been written to in this WCT while the function is called.

Parameters

<i>base</i>	WDOG peripheral base address
<i>windowValue</i>	WDOG window value.

### 35.6.13 static void WDOG\_Unlock ( WDOG\_Type \* *base* ) [inline], [static]

This function unlocks the WDOG register written. Before starting the unlock sequence and following configuration, disable the global interrupts. Otherwise, an interrupt may invalidate the unlocking sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

Parameters

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

### 35.6.14 void WDOG\_Refresh ( WDOG\_Type \* *base* )

This function feeds the WDOG. This function should be called before the WDOG timer is in timeout. Otherwise, a reset is asserted.

Parameters

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

**35.6.15 static uint16\_t WDOG\_GetResetCount( WDOG\_Type \* *base* ) [inline],  
[static]**

This function gets the WDOG reset count value.

Parameters

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

Returns

WDOG reset count value.

### 35.6.16 static void WDOG\_ClearResetCount( WDOG\_Type \* *base* ) [inline], [static]

This function clears the WDOG reset count value.

Parameters

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

# Chapter 36

## XBAR: Inter-Peripheral Crossbar Switch

### 36.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Peripheral Crossbar Switch (XBAR) block of MCUXpresso SDK devices.

The XBAR peripheral driver configures the XBAR (Inter-Peripheral Crossbar Switch) and handles initialization and configuration of the XBAR module.

XBAR driver has two parts:

- Signal connection - This part interconnects input and output signals.
- Active edge feature - Some of the outputs provides 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 bit.

### 36.2 Function groups

#### 36.2.1 XBAR Initialization

To initialize the XBAR 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 XBAR module is initialized by calling the [XBAR\\_Init\(\)](#) function.

#### 36.2.2 Call diagram

1. Call the "XBAR\_Init()" function to initialize the XBAR module.
2. Optionally, call the "XBAR\_SetSignalsConnection()" function to Set connection between the selected XBAR\_IN[\*] input and the XBAR\_OUT[\*] output signal. It connects the XBAR input to the selected XBAR output. A configuration structure of the "xbar\_input\_signal\_t" type and "xbar\_output\_signal\_t" type is required.
3. Call the "XBAR\_SetOutputSignalConfig" function to set the active edge features, such interrupts or DMA requests. A configuration structure of the "xbar\_control\_config\_t" type is required to point to structure that keeps configuration of control register.
4. Finally, the XBAR works properly.

### 36.3 Typical use case

#### Data Structures

- struct [xbar\\_control\\_config\\_t](#)  
*Defines the configuration structure of the XBAR control register. [More...](#)*

## Enumerations

- enum `xbar_active_edge_t` {
   
    `kXBAR_EdgeNone` = 0U,
   
    `kXBAR_EdgeRising` = 1U,
   
    `kXBAR_EdgeFalling` = 2U,
   
    `kXBAR_EdgeRisingAndFalling` = 3U }
   
    *XBAR active edge for detection.*
- enum `xbar_request_t` {
   
    `kXBAR_RequestDisable` = 0U,
   
    `kXBAR_RequestDMAEnable` = 1U,
   
    `kXBAR_RequestInterruptEnable` = 2U }
   
    *Defines the XBAR DMA and interrupt configurations.*
- enum `xbar_status_flag_t` {
   
    `kXBAR_EdgeDetectionOut0`,
   
    `kXBAR_EdgeDetectionOut1`,
   
    `kXBAR_EdgeDetectionOut2`,
   
    `kXBAR_EdgeDetectionOut3` }
   
    *XBAR status flags.*

## XBAR functional Operation

- void `XBAR_Init` (XBAR\_Type \*base)
   
    *Initializes the XBAR modules.*
- void `XBAR_Deinit` (XBAR\_Type \*base)
   
    *Shutdown the XBAR modules.*
- void `XBAR_SetSignalsConnection` (XBAR\_Type \*base, xbar\_input\_signal\_t input, xbar\_output\_signal\_t output)
   
    *Set connection between the selected XBAR\_IN[\*] input and the XBAR\_OUT[\*] output signal.*
- void `XBAR_ClearStatusFlags` (XBAR\_Type \*base, uint32\_t mask)
   
    *Clears the edge detection status flags of relative mask.*
- uint32\_t `XBAR_GetStatusFlags` (XBAR\_Type \*base)
   
    *Gets the active edge detection status.*
- void `XBAR_SetOutputSignalConfig` (XBAR\_Type \*base, xbar\_output\_signal\_t output, const `xbar_control_config_t` \*controlConfig)
   
    *Configures the XBAR control register.*

## 36.4 Data Structure Documentation

### 36.4.1 struct xbar\_control\_config\_t

This structure keeps the configuration of XBAR control register for one output. Control registers are available only for a few outputs. Not every XBAR module has control registers.

#### Data Fields

- `xbar_active_edge_t activeEdge`

- **xbar\_request\_t requestType**  
Selects DMA/Interrupt request.

## Field Documentation

- (1) **xbar\_active\_edge\_t xbar\_control\_config\_t::activeEdge**
- (2) **xbar\_request\_t xbar\_control\_config\_t::requestType**

## 36.5 Enumeration Type Documentation

### 36.5.1 enum xbar\_active\_edge\_t

Enumerator

**kXBAR\_EdgeNone** Edge detection status bit never asserts.

**kXBAR\_EdgeRising** Edge detection status bit asserts on rising edges.

**kXBAR\_EdgeFalling** Edge detection status bit asserts on falling edges.

**kXBAR\_EdgeRisingAndFalling** Edge detection status bit asserts on rising and falling edges.

### 36.5.2 enum xbar\_request\_t

Enumerator

**kXBAR\_RequestDisable** Interrupt and DMA are disabled.

**kXBAR\_RequestDMAEnable** DMA enabled, interrupt disabled.

**kXBAR\_RequestInterruptEnable** Interrupt enabled, DMA disabled.

### 36.5.3 enum xbar\_status\_flag\_t

This provides constants for the XBAR status flags for use in the XBAR functions.

Enumerator

**kXBAR\_EdgeDetectionOut0** XBAR\_OUT0 active edge interrupt flag, sets when active edge detected.

**kXBAR\_EdgeDetectionOut1** XBAR\_OUT1 active edge interrupt flag, sets when active edge detected.

**kXBAR\_EdgeDetectionOut2** XBAR\_OUT2 active edge interrupt flag, sets when active edge detected.

**kXBAR\_EdgeDetectionOut3** XBAR\_OUT3 active edge interrupt flag, sets when active edge detected.

## 36.6 Function Documentation

### 36.6.1 void XBAR\_Init ( **XBAR\_Type** \* *base* )

This function un-gates the XBAR clock.

Parameters

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

### 36.6.2 void XBAR\_Deinit( XBAR\_Type \* *base* )

This function disables XBAR clock.

Parameters

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

### 36.6.3 void XBAR\_SetSignalsConnection ( XBAR\_Type \* *base*, xbar\_input\_signal\_t *input*, xbar\_output\_signal\_t *output* )

This function connects the XBAR input to the selected XBAR output. If more than one XBAR module is available, only the inputs and outputs from the same module can be connected.

Example:

```
XBAR_SetSignalsConnection(XBAR, kXBARTimer_CH0_Output, kXBARDMA_INT2
);
```

Parameters

<i>base</i>	XBAR peripheral address
<i>input</i>	XBAR input signal.
<i>output</i>	XBAR output signal.

### 36.6.4 void XBAR\_ClearStatusFlags ( XBAR\_Type \* *base*, uint32\_t *mask* )

Parameters

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

<i>mask</i>	the status flags to clear.
-------------	----------------------------

### 36.6.5 `uint32_t XBAR_GetStatusFlags ( XBAR_Type * base )`

This function gets the active edge detect status of all XBAR\_OUTs. If the active edge occurs, the return value is asserted. When the interrupt or the DMA functionality is enabled for the XBAR\_OUTx, this field is 1 when the interrupt or DMA request is asserted and 0 when the interrupt or DMA request has been cleared.

Example:

```
uint32_t status;
status = XBAR_GetStatusFlags (XBAR);
```

Parameters

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

Returns

the mask of these status flag bits.

### 36.6.6 `void XBAR_SetOutputSignalConfig ( XBAR_Type * base, xbar_output_signal_t output, const xbar_control_config_t * controlConfig )`

This function configures an XBAR control register. The active edge detection and the DMA/IRQ function on the corresponding XBAR output can be set.

Example:

```
xbar_control_config_t userConfig;
userConfig.activeEdge = kXBAR_EdgeRising;
userConfig.requestType = kXBAR_RequestInterruptEnable;
XBAR_SetOutputSignalConfig (XBAR, kXBAR_OutputXB_DMA_INT0, &userConfig);
```

Parameters

<i>base</i>	XBAR peripheral address
<i>output</i>	XBAR output number.
<i>controlConfig</i>	Pointer to structure that keeps configuration of control register.

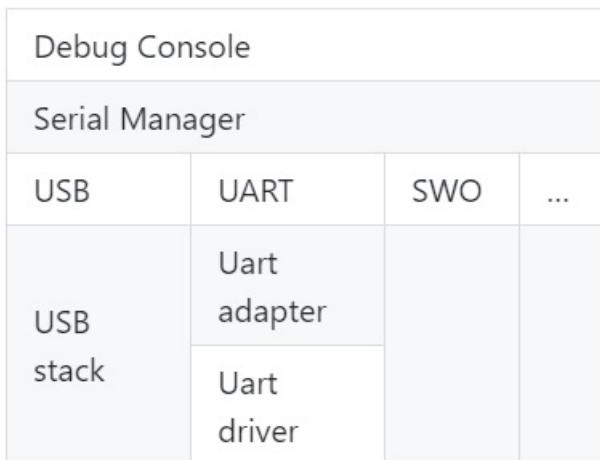
# Chapter 37

## Debug Console

### 37.1 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**

### 37.2 Function groups

#### 37.2.1 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);
```

### 37.2.2 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 */

```

### 37.2.3 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_readc` will be used when `_REDLIB_` is defined; `_write` and `_read` will be used in other cases. The macro does not specifically refer to the 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.

### 37.3 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

- 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 StrFormatPrint (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.*

## 37.4 Macro Definition Documentation

### 37.4.1 #define DEBUGCONSOLE\_REDIRECT\_TO\_TOOLCHAIN 0U

Select toolchain printf and scanf.

### 37.4.2 #define DEBUGCONSOLE\_REDIRECT\_TO\_SDK 1U

### 37.4.3 #define DEBUGCONSOLE\_DISABLE 2U

### 37.4.4 #define SDK\_DEBUGCONSOLE DEBUGCONSOLE\_REDIRECT\_TO\_SDK

The macro only support to be redefined in project setting.

### 37.4.5 #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.

## 37.5 Function Documentation

### 37.5.1 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"> <li>• kSerialPort_Uart,</li> <li>• kSerialPort_UsbCdc</li> </ul>
<i>clkSrcFreq</i>	Frequency of peripheral source clock.

Returns

Indicates whether initialization was successful or not.

Return values

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

### 37.5.2 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.

### 37.5.3 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.

### 37.5.4 status\_t DbgConsole\_ExitLowpower ( void )

This function is used to restore from low power consumption.

Returns

Indicates whether de-initialization was successful or not.

### 37.5.5 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.

### 37.5.6 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.

### 37.5.7 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.

### 37.5.8 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.

### 37.5.9 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.

**37.5.10 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.

**37.5.11 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.

**37.5.12 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.

**37.5.13 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

**37.5.14 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 "".
---------------	-----------------------------------

## 37.6 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.

### 37.6.1 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.

### 37.6.2 Guide Semihosting for Keil µVision

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

### 37.6.3 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.

### 37.6.4 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.

# Chapter 38

## Notification Framework

### 38.1 Overview

This section describes the programming interface of the Notifier driver.

### 38.2 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.*

## 38.3 Data Structure Documentation

### 38.3.1 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**

### 38.3.2 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**

### 38.3.3 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`

## 38.4 Typedef Documentation

### 38.4.1 `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.

### 38.4.2 `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.

### 38.4.3 **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.

## 38.5 Enumeration Type Documentation

### 38.5.1 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.

### 38.5.2 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.

### 38.5.3 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.

### 38.5.4 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.

## 38.6 Function Documentation

38.6.1 **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.

### 38.6.2 **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.

### 38.6.3 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 39

## Shell

### 39.1 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.

### 39.2 Function groups

#### 39.2.1 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>");
```

#### 39.2.2 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.

#### 39.2.3 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.*

## 39.3 Data Structure Documentation

### 39.3.1 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`****39.4 Macro Definition Documentation****39.4.1 `#define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE`****39.4.2 `#define SHELL_AUTO_COMPLETE (1U)`****39.4.3 `#define SHELL_BUFFER_SIZE (64U)`****39.4.4 `#define SHELL_MAX_ARGS (8U)`****39.4.5 `#define SHELL_HISTORY_COUNT (3U)`****39.4.6 `#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

**39.4.7 `#define SHELL_USE_COMMON_TASK (0U)`****39.4.8 `#define SHELL_TASK_PRIORITY (2U)`****39.4.9 `#define SHELL_TASK_STACK_SIZE (1000U)`**

### 39.4.10 #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.
-------------	--------------------------------------

### 39.4.11 #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.

### 39.4.12 #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".
----------------	--

## 39.5 Typedef Documentation

### 39.5.1 typedef shell\_status\_t(\* cmd\_function\_t)(shell\_handle\_t shellHandle, int32\_t argc, char \*\*argv)

## 39.6 Enumeration Type Documentation

### 39.6.1 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.

## 39.7 Function Documentation

### 39.7.1 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 <b>SHELL_HANDLE_SIZE</b> 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: <b>SHELL_HANDLE_DEFINE(shellHandle)</b> ; 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.

### 39.7.2 **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.

### 39.7.3 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.
------------------------------	--------------------------------------

### 39.7.4 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.

### 39.7.5 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.

### 39.7.6 **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.

### 39.7.7 **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.

### 39.7.8 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.

### 39.7.9 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.

### 39.7.10 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.
--------------------	----------------------------------

### **39.7.11 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 40

## Serial Manager

### 40.1 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 Uart

### Data Structures

- struct `serial_manager_config_t`  
*serial manager config structure. [More...](#)*
- struct `serial_manager_callback_message_t`  
*Callback message structure. [More...](#)*

### Macros

- #define `SERIAL_MANAGER_NON_BLOCKING_MODE` (0U)  
*Enable or disable serial manager non-blocking mode (1 - enable, 0 - disable)*
- #define `SERIAL_MANAGER_RING_BUFFER_FLOWCONTROL` (0U)  
*Enable or ring buffer flow control (1 - enable, 0 - disable)*
- #define `SERIAL_PORT_TYPE_UART` (0U)  
*Enable or disable uart port (1 - enable, 0 - disable)*
- #define `SERIAL_PORT_TYPE_UART_DMA` (0U)  
*Enable or disable uart dma port (1 - enable, 0 - disable)*
- #define `SERIAL_PORT_TYPE_USBCDC` (0U)  
*Enable or disable USB CDC port (1 - enable, 0 - disable)*
- #define `SERIAL_PORT_TYPE_SWO` (0U)  
*Enable or disable SWO port (1 - enable, 0 - disable)*
- #define `SERIAL_PORT_TYPE_VIRTUAL` (0U)  
*Enable or disable USB CDC virtual port (1 - enable, 0 - disable)*
- #define `SERIAL_PORT_TYPE_RPMSG` (0U)  
*Enable or disable rpmsg port (1 - enable, 0 - disable)*
- #define `SERIAL_PORT_TYPE_SPI_MASTER` (0U)  
*Enable or disable SPI Master port (1 - enable, 0 - disable)*
- #define `SERIAL_PORT_TYPE_SPI_SLAVE` (0U)  
*Enable or disable SPI Slave port (1 - enable, 0 - disable)*
- #define `SERIAL_MANAGER_TASK_HANDLE_TX` (0U)  
*Enable or disable SerialManager\_Task() handle TX to prevent recursive calling.*
- #define `SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE` (1U)  
*Set the default delay time in ms used by SerialManager\_WriteTimeDelay().*

- #define **SERIAL\_MANAGER\_READ\_TIME\_DELAY\_DEFAULT\_VALUE** (1U)  
*Set the default delay time in ms used by SerialManager\_ReadTimeDelay().*
- #define **SERIAL\_MANAGER\_TASK\_HANDLE\_RX\_AVAILABLE\_NOTIFY** (0U)  
*Enable or disable SerialManager\_Task() handle RX data available notify.*
- #define **SERIAL\_MANAGER\_WRITE\_HANDLE\_SIZE** (4U)  
*Set serial manager write handle size.*
- #define **SERIAL\_MANAGER\_USE\_COMMON\_TASK** (0U)  
*SERIAL\_PORT\_UART\_HANDLE\_SIZE/SERIAL\_PORT\_USB\_CDC\_HANDLE\_SIZE + serial manager dedicated size.*
- #define **SERIAL\_MANAGER\_HANDLE\_SIZE** (SERIAL\_MANAGER\_HANDLE\_SIZE\_TEMP + 12U)  
*Definition of serial manager handle size.*
- #define **SERIAL\_MANAGER\_HANDLE\_DEFINE**(name) uint32\_t name[((**SERIAL\_MANAGER\_HANDLE\_SIZE** + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t))]  
*Defines the serial manager handle.*
- #define **SERIAL\_MANAGER\_WRITE\_HANDLE\_DEFINE**(name) uint32\_t name[((**SERIAL\_MANAGER\_WRITE\_HANDLE\_SIZE** + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t))]  
*Defines the serial manager write handle.*
- #define **SERIAL\_MANAGER\_READ\_HANDLE\_DEFINE**(name) uint32\_t name[((**SERIAL\_MANAGER\_READ\_HANDLE\_SIZE** + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t))]  
*Defines the serial manager read handle.*
- #define **SERIAL\_MANAGER\_TASK\_PRIORITY** (2U)  
*Macro to set serial manager task priority.*
- #define **SERIAL\_MANAGER\_TASK\_STACK\_SIZE** (1000U)  
*Macro to set serial manager task stack size.*

## Typedefs

- typedef void \* **serial\_handle\_t**  
*The handle of the serial manager module.*
- typedef void \* **serial\_write\_handle\_t**  
*The write handle of the serial manager module.*
- typedef void \* **serial\_read\_handle\_t**  
*The read handle of the serial manager module.*
- typedef void(\* **serial\_manager\_callback\_t** )(void \*callbackParam, **serial\_manager\_callback\_message\_t** \*message, **serial\_manager\_status\_t** status)  
*serial manager callback function*
- typedef void(\* **serial\_manager\_lowpower\_critical\_callback\_t** )(void)  
*serial manager Lowpower Critical callback function*

## Enumerations

- enum `serial_port_type_t` {
   
  `kSerialPort_None` = 0U,
   
  `kSerialPort_Uart` = 1U,
   
  `kSerialPort_UsbCdc`,
   
  `kSerialPort_Swo`,
   
  `kSerialPort_Virtual`,
   
  `kSerialPort_Rpmsg`,
   
  `kSerialPort_UartDma`,
   
  `kSerialPort_SpiMaster`,
   
  `kSerialPort_SpiSlave` }
   
    *serial port type*
- enum `serial_manager_type_t` {
   
  `kSerialManager_NonBlocking` = 0x0U,
   
  `kSerialManager_Blocking` = 0x8F41U }
   
    *serial manager type*
- enum `serial_manager_status_t` {
   
  `kStatus_SerialManager_Success` = `kStatus_Success`,
   
  `kStatus_SerialManager_Error` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 1)`,
   
  `kStatus_SerialManager_Busy` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 2)`,
   
  `kStatus_SerialManager_Notify` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 3)`,
   
  `kStatus_SerialManager_Canceled`,
   
  `kStatus_SerialManager_HandleConflict` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 5)`,
   
  `kStatus_SerialManager_RingBufferOverflow`,
   
  `kStatus_SerialManager_NotConnected` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 7)` }
   
    *serial manager error code*

## Functions

- `serial_manager_status_t SerialManager_Init (serial_handle_t serialHandle, const serial_manager_config_t *config)`

*Initializes a serial manager module with the serial manager handle and the user configuration structure.*
- `serial_manager_status_t SerialManager_Deinit (serial_handle_t serialHandle)`

*De-initializes the serial manager module instance.*
- `serial_manager_status_t SerialManager_OpenWriteHandle (serial_handle_t serialHandle, serial_write_handle_t writeHandle)`

*Opens a writing handle for the serial manager module.*
- `serial_manager_status_t SerialManager_CloseWriteHandle (serial_write_handle_t writeHandle)`

*Closes a writing handle for the serial manager module.*
- `serial_manager_status_t SerialManager_OpenReadHandle (serial_handle_t serialHandle, serial_read_handle_t readHandle)`

*Opens a reading handle for the serial manager module.*
- `serial_manager_status_t SerialManager_CloseReadHandle (serial_read_handle_t readHandle)`

*Closes a reading for the serial manager module.*

- `serial_manager_status_t SerialManager_WriteBlocking (serial_write_handle_t writeHandle, uint8_t *buffer, uint32_t length)`  
*Transmits data with the blocking mode.*
- `serial_manager_status_t SerialManager_ReadBlocking (serial_read_handle_t readHandle, uint8_t *buffer, uint32_t length)`  
*Reads data with the blocking mode.*
- `serial_manager_status_t SerialManager_EnterLowpower (serial_handle_t serialHandle)`  
*Prepares to enter low power consumption.*
- `serial_manager_status_t SerialManager_ExitLowpower (serial_handle_t serialHandle)`  
*Restores from low power consumption.*
- `void SerialManager_SetLowpowerCriticalCb (const serial_manager_lowpower_critical_CBs_t *pfCallback)`  
*This function performs initialization of the callbacks structure used to disable lowpower when serial manager is active.*

## 40.2 Data Structure Documentation

### 40.2.1 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.

### 40.2.2 struct serial\_manager\_callback\_message\_t

#### Data Fields

- `uint8_t * buffer`  
*Transferred buffer.*
- `uint32_t length`  
*Transferred data length.*

## 40.3 Macro Definition Documentation

**40.3.1 #define SERIAL\_MANAGER\_WRITE\_TIME\_DELAY\_DEFAULT\_VALUE (1U)**

**40.3.2 #define SERIAL\_MANAGER\_READ\_TIME\_DELAY\_DEFAULT\_VALUE (1U)**

**40.3.3 #define SERIAL\_MANAGER\_USE\_COMMON\_TASK (0U)**

Macro to determine whether use common task.

**40.3.4 #define SERIAL\_MANAGER\_HANDLE\_SIZE (SERIAL\_MANAGER\_HANDLE\_SIZE\_TEMP + 12U)**

**40.3.5 #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.
-------------	---

**40.3.6 #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.
-------------	---

#### 40.3.7 #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.
-------------	--

#### 40.3.8 #define SERIAL\_MANAGER\_TASK\_PRIORITY (2U)

#### 40.3.9 #define SERIAL\_MANAGER\_TASK\_STACK\_SIZE (1000U)

### 40.4 Enumeration Type Documentation

#### 40.4.1 enum serial\_port\_type\_t

Enumerator

- kSerialPort\_None* Serial port is none.
- kSerialPort\_Uart* Serial port UART.
- kSerialPort\_UsbCdc* Serial port USB CDC.
- kSerialPort\_Swo* Serial port SWO.
- kSerialPort\_Virtual* Serial port Virtual.
- kSerialPort\_Rpmsg* Serial port RPMSG.
- kSerialPort\_UartDma* Serial port UART DMA.

*kSerialPort\_SpiMaster* Serial port SPIMASTER.

*kSerialPort\_SpiSlave* Serial port SPISLAVE.

#### 40.4.2 enum serial\_manager\_type\_t

Enumerator

*kSerialManager\_NonBlocking* None blocking handle.

*kSerialManager\_Blocking* Blocking handle.

#### 40.4.3 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.

### 40.5 Function Documentation

#### 40.5.1 serial\_manager\_status\_t SerialManager\_Init ( serial\_handle\_t *serialHandle*, const serial\_manager\_config\_t \* *config* )

This function configures the Serial Manager module with user-defined settings. The user can configure the configuration structure. The parameter *serialHandle* is a pointer to point to a memory space of size [SERIAL\\_MANAGER\\_HANDLE\\_SIZE](#) allocated by the caller. The Serial Manager module supports three types of serial port, UART (includes UART, USART, LPSCI, LPUART, etc), USB CDC and swo. Please refer to [serial\\_port\\_type\\_t](#) for serial port setting. These three types can be set by using [serial\\_manager\\_config\\_t](#).

Example below shows how to use this API to configure the Serial Manager. For UART,

```
* #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
* static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
* static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
* serial_manager_config_t config;
* serial_port_uart_config_t uartConfig;
* config.type = kSerialPort_Uart;
* config.ringBuffer = &s_ringBuffer[0];
* config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
* uartConfig.instance = 0;
```

```

*   uartConfig.clockRate = 24000000;
*   uartConfig.baudRate = 115200;
*   uartConfig.parityMode = kSerialManager_UartParityDisabled;
*   uartConfig.stopBitCount = kSerialManager_UartOneStopBit;
*   uartConfig.enableRx = 1;
*   uartConfig.enableTx = 1;
*   uartConfig.enableRxRTS = 0;
*   uartConfig.enableTxCTS = 0;
*   config.portConfig = &uartConfig;
*   SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*

```

For USB CDC,

```

*   #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
*   static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
*   static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
*   serial_manager_config_t config;
*   serial_port_usb_cdc_config_t usbCdcConfig;
*   config.type = kSerialPort_UsbCdc;
*   config.ringBuffer = &s_ringBuffer[0];
*   config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
*   usbCdcConfig.controllerIndex = kSerialManager_UsbControllerKhci0;
*   config.portConfig = &usbCdcConfig;
*   SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*

```

Parameters

<i>serialHandle</i>	Pointer to point to a memory space of size <a href="#">SERIAL_MANAGER_HANDLE_SIZE</a> allocated by the caller. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: <a href="#">SERIAL_MANAGER_HANDLE_DEFINE(serialHandle)</a> ; or <code>uint32_t serialHandle[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>
<i>config</i>	Pointer to user-defined configuration structure.

Return values

<i>kStatus_SerialManager_Error</i>	An error occurred.
<i>kStatus_SerialManager_Success</i>	The Serial Manager module initialization succeed.

#### 40.5.2 **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.

#### 40.5.3 **serial\_manager\_status\_t SerialManager\_OpenWriteHandle ( serial\_handle\_t *serialHandle*, serial\_write\_handle\_t *writeHandle* )**

This function Opens a writing handle for the serial manager module. If the serial manager needs to be used in different tasks, the task should open a dedicated write handle for itself by calling [SerialManager\\_OpenWriteHandle](#). Since there can only one buffer for transmission for the writing handle at the same time, multiple writing handles need to be opened when the multiple transmission is needed for a task.

## Parameters

<i>serialHandle</i>	The serial manager module handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.
<i>writeHandle</i>	The serial manager module writing handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: <a href="#">SERIAL_MANAGER_WRITE_HANDLE_DEFINE(writeHandle)</a> ; or <code>uint32_t writeHandle[((SERIAL_MANAGER_WRITE_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>

## Return values

<i>kStatus_SerialManager_-Error</i>	An error occurred.
<i>kStatus_SerialManager_-HandleConflict</i>	The writing handle was opened.

<i>kStatus_SerialManager_-Success</i>	The writing handle is opened.
---------------------------------------	-------------------------------

Example below shows how to use this API to write data. For task 1,

```
* static SERIAL_MANAGER_WRITE_HANDLE_DEFINE(s_serialWriteHandle1);
* static uint8_t s_nonBlockingWelcome1[] = "This is non-blocking writing log for task1!\r\n";
* SerialManager_OpenWriteHandle((serial_handle_t)serialHandle
, (serial_write_handle_t)s_serialWriteHandle1);
* SerialManager_InstallTxCallback((serial_write_handle_t)s_serialWriteHandle1,
, Task1_SerialManagerTxCallback,
s_serialWriteHandle1);
* SerialManager_WriteNonBlocking((serial_write_handle_t)s_serialWriteHandle1,
s_nonBlockingWelcome1,
sizeof(s_nonBlockingWelcome1) - 1U);
*
```

For task 2,

```
* static SERIAL_MANAGER_WRITE_HANDLE_DEFINE(s_serialWriteHandle2);
* static uint8_t s_nonBlockingWelcome2[] = "This is non-blocking writing log for task2!\r\n";
* SerialManager_OpenWriteHandle((serial_handle_t)serialHandle
, (serial_write_handle_t)s_serialWriteHandle2);
* SerialManager_InstallTxCallback((serial_write_handle_t)s_serialWriteHandle2,
, Task2_SerialManagerTxCallback,
s_serialWriteHandle2);
* SerialManager_WriteNonBlocking((serial_write_handle_t)s_serialWriteHandle2,
s_nonBlockingWelcome2,
sizeof(s_nonBlockingWelcome2) - 1U);
*
```

#### 40.5.4 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.
---------------------------------------	-------------------------------

#### 40.5.5 serial\_manager\_status\_t SerialManager\_OpenReadHandle ( serial\_handle\_t *serialHandle*, serial\_read\_handle\_t *readHandle* )

This function Opens a reading handle for the serial manager module. The reading handle can not be opened multiple at the same time. The error code kStatus\_SerialManager\_Busy would be returned when

the previous reading handle is not closed. And there can only be one buffer for receiving for the reading handle at the same time.

## Parameters

<i>serialHandle</i>	The serial manager module handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.
<i>readHandle</i>	The serial manager module reading handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: <a href="#">SERIAL_MANAGER_READ_HANDLE_DEFINE(readHandle)</a> ; or <code>uint32_t readHandle[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>

## Return values

<i>kStatus_SerialManager_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));
*
```

#### 40.5.6 **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.
---------------------------------------	-------------------------------

#### 40.5.7 `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.

#### 40.5.8 `serial_manager_status_t SerialManager_ReadBlocking ( serial_read_handle_t readHandle, uint8_t * buffer, uint32_t length )`

This is a blocking function, which polls the receiving buffer, waits for the receiving buffer to be full. This function receives data using an interrupt method. The interrupt of the hardware could not be disabled. And There can only one buffer for receiving for the reading handle at the same time.

## Note

The function `SerialManager_ReadBlocking` and the function `SerialManager_ReadNonBlocking` cannot be used at the same time. And, the function `SerialManager_CancelReading` cannot be used to abort the transmission of this function.

## Parameters

<i>readHandle</i>	The serial manager module handle pointer.
<i>buffer</i>	Start address of the data to store the received data.
<i>length</i>	The length of the data to be received.

## Return values

<i>kStatus_SerialManager_-Success</i>	Successfully received all data.
<i>kStatus_SerialManager_-Busy</i>	Previous transmission still not finished; data not all received yet.
<i>kStatus_SerialManager_-Error</i>	An error occurred.

#### 40.5.9 `serial_manager_status_t SerialManager_EnterLowpower ( serial_handle_t serialHandle )`

This function is used to prepare to enter low power consumption.

## Parameters

<i>serialHandle</i>	The serial manager module handle pointer.
---------------------	---

## Return values

<i>kStatus_SerialManager_-Success</i>	Successful operation.
---------------------------------------	-----------------------

#### 40.5.10 `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.
---------------------------------------	-----------------------

#### **40.5.11 void SerialManager\_SetLowpowerCriticalCb ( const serial\_manager\_lowpower\_critical\_CBs\_t \* *pfCallback* )**

Parameters

<i>pfCallback</i>	Pointer to the function structure used to allow/disable lowpower.
-------------------	---

## 40.6 Serial Port Uart

### 40.6.1 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*

### 40.6.2 Enumeration Type Documentation

#### 40.6.2.1 enum serial\_port\_uart\_parity\_mode\_t

Enumerator

**kSerialManager\_UartParityDisabled** Parity disabled.  
**kSerialManager\_UartParityEven** Parity even enabled.  
**kSerialManager\_UartParityOdd** Parity odd enabled.

#### 40.6.2.2 enum serial\_port\_uart\_stop\_bit\_count\_t

Enumerator

**kSerialManager\_UartOneStopBit** One stop bit.  
**kSerialManager\_UartTwoStopBit** Two stop bits.

**How to Reach Us:**

**Home Page:**

[nxp.com](http://nxp.com)

**Web Support:**

[nxp.com/support](http://nxp.com/support)

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

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

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, Freescale, the Freescale logo, Kinetis, Processor Expert, and Tower are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, Cortex, Keil, Mbed, Mbed Enabled, and Vision are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2021 NXP B.V.

