

# Kinetis SDK v.2.0 API Reference Manual

**NXP Semiconductors**

Document Number: KSDKKL8220APIRM  
Rev. 0  
Jun 2016





# Contents

**Chapter Introduction**

**Chapter Driver errors status**

**Chapter Architectural Overview**

**Chapter Trademarks**

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

<b>5.1</b>	<b>Overview</b>	11
<b>5.2</b>	<b>Typical use case</b>	11
5.2.1	Polling Configuration	11
5.2.2	Interrupt Configuration	11
<b>5.3</b>	<b>Data Structure Documentation</b>	14
5.3.1	struct adc16_config_t	14
5.3.2	struct adc16_hardware_compare_config_t	15
5.3.3	struct adc16_channel_config_t	16
<b>5.4</b>	<b>Macro Definition Documentation</b>	16
5.4.1	FSL_ADC16_DRIVER_VERSION	16
<b>5.5</b>	<b>Enumeration Type Documentation</b>	16
5.5.1	_adc16_channel_status_flags	16
5.5.2	_adc16_status_flags	16
5.5.3	adc16_clock_divider_t	16
5.5.4	adc16_resolution_t	17
5.5.5	adc16_clock_source_t	17
5.5.6	adc16_long_sample_mode_t	17
5.5.7	adc16_reference_voltage_source_t	17
5.5.8	adc16_hardware_compare_mode_t	18
<b>5.6</b>	<b>Function Documentation</b>	18
5.6.1	ADC16_Init	18
5.6.2	ADC16_Deinit	18
5.6.3	ADC16_GetDefaultConfig	18

# Contents

Section Number	Title	Page Number
5.6.4	ADC16_EnableHardwareTrigger . . . . .	19
5.6.5	ADC16_SetHardwareCompareConfig . . . . .	19
5.6.6	ADC16_GetStatusFlags . . . . .	19
5.6.7	ADC16_ClearStatusFlags . . . . .	19
5.6.8	ADC16_SetChannelConfig . . . . .	20
5.6.9	ADC16_GetChannelConversionValue . . . . .	20
5.6.10	ADC16_GetChannelStatusFlags . . . . .	21

## Chapter Clock Driver

6.1	Overview . . . . .	23
6.2	Get frequency . . . . .	23
6.3	External clock frequency . . . . .	23
6.4	Data Structure Documentation . . . . .	31
6.4.1	struct sim_clock_config_t . . . . .	31
6.4.2	struct oscer_config_t . . . . .	31
6.4.3	struct osc_config_t . . . . .	32
6.4.4	struct mcg_pll_config_t . . . . .	32
6.4.5	struct mcg_config_t . . . . .	33
6.5	Macro Definition Documentation . . . . .	34
6.5.1	FSL_CLOCK_DRIVER_VERSION . . . . .	34
6.5.2	MCG_INTERNAL_IRC_48M . . . . .	34
6.5.3	DMAMUX_CLOCKS . . . . .	34
6.5.4	RTC_CLOCKS . . . . .	34
6.5.5	DRYICE_CLOCKS . . . . .	34
6.5.6	PORT_CLOCKS . . . . .	35
6.5.7	EWM_CLOCKS . . . . .	35
6.5.8	PIT_CLOCKS . . . . .	35
6.5.9	DSPI_CLOCKS . . . . .	35
6.5.10	EMVSIM_CLOCKS . . . . .	35
6.5.11	QSPI_CLOCKS . . . . .	36
6.5.12	EDMA_CLOCKS . . . . .	36
6.5.13	LPUART_CLOCKS . . . . .	36
6.5.14	DAC_CLOCKS . . . . .	36
6.5.15	LPTMR_CLOCKS . . . . .	36
6.5.16	ADC16_CLOCKS . . . . .	37
6.5.17	TRNG_CLOCKS . . . . .	37
6.5.18	MPU_CLOCKS . . . . .	37
6.5.19	FLEXIO_CLOCKS . . . . .	37
6.5.20	VREF_CLOCKS . . . . .	37
6.5.21	TPM_CLOCKS . . . . .	38

# Contents

Section Number	Title	Page Number
6.5.22	TSI_CLOCKS . . . . .	38
6.5.23	LTC_CLOCKS . . . . .	38
6.5.24	CRC_CLOCKS . . . . .	38
6.5.25	I2C_CLOCKS . . . . .	38
6.5.26	CMP_CLOCKS . . . . .	39
6.5.27	INTMUX_CLOCKS . . . . .	39
6.5.28	SYS_CLK . . . . .	39
<b>6.6</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>39</b>
6.6.1	clock_name_t . . . . .	39
6.6.2	clock_usb_src_t . . . . .	40
6.6.3	clock_ip_name_t . . . . .	40
6.6.4	osc_mode_t . . . . .	40
6.6.5	_osc_cap_load . . . . .	40
6.6.6	_oscer_enable_mode . . . . .	40
6.6.7	mcg_fll_src_t . . . . .	40
6.6.8	mcg_irc_mode_t . . . . .	41
6.6.9	mcg_dmx32_t . . . . .	41
6.6.10	mcg_drs_t . . . . .	41
6.6.11	mcg_pll_ref_src_t . . . . .	41
6.6.12	mcg_clkout_src_t . . . . .	41
6.6.13	mcg_atm_select_t . . . . .	42
6.6.14	mcg_oscsel_t . . . . .	42
6.6.15	mcg_pll_clk_select_t . . . . .	42
6.6.16	mcg_monitor_mode_t . . . . .	42
6.6.17	_mcg_status . . . . .	42
6.6.18	_mcg_status_flags_t . . . . .	43
6.6.19	_mcg_irclk_enable_mode . . . . .	43
6.6.20	_mcg_pll_enable_mode . . . . .	43
6.6.21	mcg_mode_t . . . . .	43
<b>6.7</b>	<b>Function Documentation</b> . . . . .	<b>44</b>
6.7.1	CLOCK_EnableClock . . . . .	44
6.7.2	CLOCK_DisableClock . . . . .	45
6.7.3	CLOCK_SetEr32kClock . . . . .	45
6.7.4	CLOCK_SetEmvsimClock . . . . .	45
6.7.5	CLOCK_SetLpuartClock . . . . .	45
6.7.6	CLOCK_SetTpmClock . . . . .	45
6.7.7	CLOCK_SetFlexio0Clock . . . . .	46
6.7.8	CLOCK_SetPLIFllSelClock . . . . .	46
6.7.9	CLOCK_SetClkOutClock . . . . .	46
6.7.10	CLOCK_SetRtcClkOutClock . . . . .	46
6.7.11	CLOCK_EnableUsbfs0Clock . . . . .	46
6.7.12	CLOCK_DisableUsbfs0Clock . . . . .	47
6.7.13	CLOCK_SetOutDiv . . . . .	47

# Contents

Section Number	Title	Page Number
6.7.14	CLOCK_GetFreq . . . . .	47
6.7.15	CLOCK_GetCoreSysClkFreq . . . . .	48
6.7.16	CLOCK_GetPlatClkFreq . . . . .	48
6.7.17	CLOCK_GetBusClkFreq . . . . .	48
6.7.18	CLOCK_GetFlashClkFreq . . . . .	48
6.7.19	CLOCK_GetPllFllSelClkFreq . . . . .	48
6.7.20	CLOCK_GetQspiBusClkFreq . . . . .	49
6.7.21	CLOCK_GetEr32kClkFreq . . . . .	49
6.7.22	CLOCK_GetOsc0ErClkUndivFreq . . . . .	49
6.7.23	CLOCK_GetOsc0ErClkFreq . . . . .	49
6.7.24	CLOCK_SetSimConfig . . . . .	49
6.7.25	CLOCK_SetSimSafeDivs . . . . .	49
6.7.26	CLOCK_GetOutClkFreq . . . . .	50
6.7.27	CLOCK_GetFllFreq . . . . .	50
6.7.28	CLOCK_GetInternalRefClkFreq . . . . .	50
6.7.29	CLOCK_GetFixedFreqClkFreq . . . . .	50
6.7.30	CLOCK_GetPll0Freq . . . . .	51
6.7.31	CLOCK_SetLowPowerEnable . . . . .	51
6.7.32	CLOCK_SetInternalRefClkConfig . . . . .	51
6.7.33	CLOCK_SetExternalRefClkConfig . . . . .	52
6.7.34	CLOCK_EnablePll0 . . . . .	52
6.7.35	CLOCK_DisablePll0 . . . . .	52
6.7.36	CLOCK_CalcPllDiv . . . . .	52
6.7.37	CLOCK_SetOsc0MonitorMode . . . . .	54
6.7.38	CLOCK_SetRtcOscMonitorMode . . . . .	54
6.7.39	CLOCK_SetPll0MonitorMode . . . . .	54
6.7.40	CLOCK_GetStatusFlags . . . . .	54
6.7.41	CLOCK_ClearStatusFlags . . . . .	55
6.7.42	OSC_SetExtRefClkConfig . . . . .	55
6.7.43	OSC_SetCapLoad . . . . .	56
6.7.44	CLOCK_InitOsc0 . . . . .	56
6.7.45	CLOCK_DeinitOsc0 . . . . .	56
6.7.46	CLOCK_SetXtal0Freq . . . . .	56
6.7.47	CLOCK_SetXtal32Freq . . . . .	57
6.7.48	CLOCK_TrimInternalRefClk . . . . .	57
6.7.49	CLOCK_GetMode . . . . .	58
6.7.50	CLOCK_SetFeiMode . . . . .	58
6.7.51	CLOCK_SetFeeMode . . . . .	58
6.7.52	CLOCK_SetFbiMode . . . . .	59
6.7.53	CLOCK_SetFbeMode . . . . .	60
6.7.54	CLOCK_SetBlpiMode . . . . .	61
6.7.55	CLOCK_SetBlpeMode . . . . .	61
6.7.56	CLOCK_SetPbeMode . . . . .	62
6.7.57	CLOCK_SetPeeMode . . . . .	63
6.7.58	CLOCK_ExternalModeToFbeModeQuick . . . . .	63

# Contents

Section Number	Title	Page Number
6.7.59	<a href="#">CLOCK_InternalModeToFbiModeQuick</a>	64
6.7.60	<a href="#">CLOCK_BootToFeiMode</a>	64
6.7.61	<a href="#">CLOCK_BootToFeeMode</a>	65
6.7.62	<a href="#">CLOCK_BootToBlpiMode</a>	65
6.7.63	<a href="#">CLOCK_BootToBlpeMode</a>	66
6.7.64	<a href="#">CLOCK_BootToPeeMode</a>	66
6.7.65	<a href="#">CLOCK_SetMcgConfig</a>	67
<b>6.8</b>	<b><a href="#">Variable Documentation</a></b>	<b>67</b>
6.8.1	<a href="#">g_xtal0Freq</a>	67
6.8.2	<a href="#">g_xtal32Freq</a>	68
<b>6.9</b>	<b><a href="#">Multipurpose Clock Generator (MCG)</a></b>	<b>69</b>
6.9.1	<a href="#">Function description</a>	69
6.9.2	<a href="#">Typical use case</a>	71
 <b>Chapter CMP: Analog Comparator Driver</b>		
<b>7.1</b>	<b><a href="#">Overview</a></b>	<b>75</b>
<b>7.2</b>	<b><a href="#">Typical use case</a></b>	<b>75</b>
7.2.1	<a href="#">Polling Configuration</a>	75
7.2.2	<a href="#">Interrupt Configuration</a>	75
<b>7.3</b>	<b><a href="#">Data Structure Documentation</a></b>	<b>78</b>
7.3.1	<a href="#">struct cmp_config_t</a>	78
7.3.2	<a href="#">struct cmp_filter_config_t</a>	78
7.3.3	<a href="#">struct cmp_dac_config_t</a>	79
<b>7.4</b>	<b><a href="#">Macro Definition Documentation</a></b>	<b>79</b>
7.4.1	<a href="#">FSL_CMP_DRIVER_VERSION</a>	79
<b>7.5</b>	<b><a href="#">Enumeration Type Documentation</a></b>	<b>79</b>
7.5.1	<a href="#">_cmp_interrupt_enable</a>	79
7.5.2	<a href="#">_cmp_status_flags</a>	79
7.5.3	<a href="#">cmp_hysteresis_mode_t</a>	80
7.5.4	<a href="#">cmp_reference_voltage_source_t</a>	80
<b>7.6</b>	<b><a href="#">Function Documentation</a></b>	<b>80</b>
7.6.1	<a href="#">CMP_Init</a>	80
7.6.2	<a href="#">CMP_Deinit</a>	80
7.6.3	<a href="#">CMP_Enable</a>	82
7.6.4	<a href="#">CMP_GetDefaultConfig</a>	82
7.6.5	<a href="#">CMP_SetInputChannels</a>	82
7.6.6	<a href="#">CMP_SetFilterConfig</a>	83
7.6.7	<a href="#">CMP_SetDACCConfig</a>	83

# Contents

Section Number	Title	Page Number
7.6.8	CMP_EnableInterrupts . . . . .	83
7.6.9	CMP_DisableInterrupts . . . . .	83
7.6.10	CMP_GetStatusFlags . . . . .	84
7.6.11	CMP_ClearStatusFlags . . . . .	84
<b>Chapter</b>	<b>CRC: Cyclic Redundancy Check Driver</b>	
8.1	<b>Overview</b> . . . . .	85
8.2	<b>CRC Driver Initialization and Configuration</b> . . . . .	85
8.3	<b>CRC Write Data</b> . . . . .	85
8.4	<b>CRC Get Checksum</b> . . . . .	85
8.5	<b>Comments about API usage in RTOS</b> . . . . .	86
8.6	<b>Comments about API usage in interrupt handler</b> . . . . .	86
8.7	<b>CRC Driver Examples</b> . . . . .	86
8.7.1	Simple examples . . . . .	86
8.7.2	Advanced examples . . . . .	87
8.8	<b>Data Structure Documentation</b> . . . . .	90
8.8.1	struct crc_config_t . . . . .	90
8.9	<b>Macro Definition Documentation</b> . . . . .	90
8.9.1	FSL_CRC_DRIVER_VERSION . . . . .	90
8.9.2	CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT . . . . .	91
8.10	<b>Enumeration Type Documentation</b> . . . . .	91
8.10.1	crc_bits_t . . . . .	91
8.10.2	crc_result_t . . . . .	91
8.11	<b>Function Documentation</b> . . . . .	91
8.11.1	CRC_Init . . . . .	91
8.11.2	CRC_Deinit . . . . .	91
8.11.3	CRC_GetDefaultConfig . . . . .	92
8.11.4	CRC_WriteData . . . . .	92
8.11.5	CRC_Get32bitResult . . . . .	92
8.11.6	CRC_Get16bitResult . . . . .	93
<b>Chapter</b>	<b>DAC: Digital-to-Analog Converter Driver</b>	
9.1	<b>Overview</b> . . . . .	95
9.2	<b>Typical use case</b> . . . . .	95

Section Number	Title	Page Number
9.2.1	Working as a basic DAC without the hardware buffer feature. . . . .	95
9.2.2	Working with the hardware buffer. . . . .	95
<b>9.3</b>	<b>Data Structure Documentation</b> . . . . .	<b>98</b>
9.3.1	<b>struct dac_config_t</b> . . . . .	98
9.3.2	<b>struct dac_buffer_config_t</b> . . . . .	98
<b>9.4</b>	<b>Macro Definition Documentation</b> . . . . .	<b>99</b>
9.4.1	<b>FSL_DAC_DRIVER_VERSION</b> . . . . .	99
<b>9.5</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>99</b>
9.5.1	<b>_dac_buffer_status_flags</b> . . . . .	99
9.5.2	<b>_dac_buffer_interrupt_enable</b> . . . . .	99
9.5.3	<b>dac_reference_voltage_source_t</b> . . . . .	99
9.5.4	<b>dac_buffer_trigger_mode_t</b> . . . . .	99
9.5.5	<b>dac_buffer_work_mode_t</b> . . . . .	99
<b>9.6</b>	<b>Function Documentation</b> . . . . .	<b>100</b>
9.6.1	<b>DAC_Init</b> . . . . .	100
9.6.2	<b>DAC_Deinit</b> . . . . .	100
9.6.3	<b>DAC_GetDefaultConfig</b> . . . . .	100
9.6.4	<b>DAC_Enable</b> . . . . .	100
9.6.5	<b>DAC_EnableBuffer</b> . . . . .	101
9.6.6	<b>DAC_SetBufferConfig</b> . . . . .	101
9.6.7	<b>DAC_GetDefaultBufferConfig</b> . . . . .	101
9.6.8	<b>DAC_EnableBufferDMA</b> . . . . .	101
9.6.9	<b>DAC_SetBufferValue</b> . . . . .	102
9.6.10	<b>DAC_DoSoftwareTriggerBuffer</b> . . . . .	102
9.6.11	<b>DAC_GetBufferReadPointer</b> . . . . .	102
9.6.12	<b>DAC_SetBufferReadPointer</b> . . . . .	103
9.6.13	<b>DAC_EnableBufferInterrupts</b> . . . . .	103
9.6.14	<b>DAC_DisableBufferInterrupts</b> . . . . .	103
9.6.15	<b>DAC_GetBufferStatusFlags</b> . . . . .	103
9.6.16	<b>DAC_ClearBufferStatusFlags</b> . . . . .	103

## Chapter **DMAMUX: Direct Memory Access Multiplexer Driver**

<b>10.1</b>	<b>Overview</b> . . . . .	<b>105</b>
<b>10.2</b>	<b>Typical use case</b> . . . . .	<b>105</b>
10.2.1	<b>DMAMUX Operation</b> . . . . .	105
<b>10.3</b>	<b>Macro Definition Documentation</b> . . . . .	<b>105</b>
10.3.1	<b>FSL_DMAMUX_DRIVER_VERSION</b> . . . . .	105
<b>10.4</b>	<b>Function Documentation</b> . . . . .	<b>106</b>

# Contents

Section Number	Title	Page Number
10.4.1	DMAMUX_Init . . . . .	106
10.4.2	DMAMUX_Deinit . . . . .	107
10.4.3	DMAMUX_EnableChannel . . . . .	107
10.4.4	DMAMUX_DisableChannel . . . . .	107
10.4.5	DMAMUX_SetSource . . . . .	108
<b>Chapter DSPI: Serial Peripheral Interface Driver</b>		
<b>11.1</b>	<b>Overview . . . . .</b>	<b>109</b>
<b>11.2</b>	<b>DSPI Driver . . . . .</b>	<b>110</b>
11.2.1	Overview . . . . .	110
11.2.2	Typical use case . . . . .	110
11.2.3	Data Structure Documentation . . . . .	117
11.2.4	Macro Definition Documentation . . . . .	124
11.2.5	Typedef Documentation . . . . .	125
11.2.6	Enumeration Type Documentation . . . . .	126
11.2.7	Function Documentation . . . . .	130
<b>11.3</b>	<b>DSPI DMA Driver . . . . .</b>	<b>150</b>
11.3.1	Overview . . . . .	150
11.3.2	Data Structure Documentation . . . . .	151
11.3.3	Typedef Documentation . . . . .	154
11.3.4	Function Documentation . . . . .	155
<b>11.4</b>	<b>DSPI eDMA Driver . . . . .</b>	<b>160</b>
11.4.1	Overview . . . . .	160
11.4.2	Data Structure Documentation . . . . .	161
11.4.3	Typedef Documentation . . . . .	164
11.4.4	Function Documentation . . . . .	165
<b>11.5</b>	<b>DSPI FreeRTOS Driver . . . . .</b>	<b>170</b>
11.5.1	Overview . . . . .	170
11.5.2	Data Structure Documentation . . . . .	170
11.5.3	Function Documentation . . . . .	171
<b>11.6</b>	<b>DSPI μCOS/II Driver . . . . .</b>	<b>173</b>
11.6.1	Overview . . . . .	173
11.6.2	Data Structure Documentation . . . . .	173
11.6.3	Function Documentation . . . . .	174
<b>11.7</b>	<b>DSPI μCOS/III Driver . . . . .</b>	<b>176</b>
11.7.1	Overview . . . . .	176
11.7.2	Data Structure Documentation . . . . .	176
11.7.3	Function Documentation . . . . .	177

# Contents

Section Number	Title	Page Number
Chapter	eDMA: Enhanced Direct Memory Access Controller (eDMA) Driver	
12.1	Overview . . . . .	179
12.2	Typical use case . . . . .	179
12.2.1	eDMA Operation . . . . .	179
12.3	Data Structure Documentation . . . . .	185
12.3.1	struct edma_config_t . . . . .	185
12.3.2	struct edma_transfer_config_t . . . . .	185
12.3.3	struct edma_channel_Preemption_config_t . . . . .	186
12.3.4	struct edma_minor_offset_config_t . . . . .	187
12.3.5	struct edma_tcd_t . . . . .	187
12.3.6	struct edma_handle_t . . . . .	188
12.4	Macro Definition Documentation . . . . .	189
12.4.1	FSL_EDMA_DRIVER_VERSION . . . . .	189
12.5	Typedef Documentation . . . . .	189
12.5.1	edma_callback . . . . .	189
12.6	Enumeration Type Documentation . . . . .	189
12.6.1	edma_transfer_size_t . . . . .	189
12.6.2	edma_modulo_t . . . . .	190
12.6.3	edma_bandwidth_t . . . . .	190
12.6.4	edma_channel_link_type_t . . . . .	191
12.6.5	_edma_channel_status_flags . . . . .	191
12.6.6	_edma_error_status_flags . . . . .	191
12.6.7	edma_interrupt_enable_t . . . . .	191
12.6.8	edma_transfer_type_t . . . . .	192
12.6.9	_edma_transfer_status . . . . .	192
12.7	Function Documentation . . . . .	192
12.7.1	EDMA_Init . . . . .	192
12.7.2	EDMA_Deinit . . . . .	192
12.7.3	EDMA_GetDefaultConfig . . . . .	192
12.7.4	EDMA_ResetChannel . . . . .	193
12.7.5	EDMA_SetTransferConfig . . . . .	193
12.7.6	EDMA_SetMinorOffsetConfig . . . . .	194
12.7.7	EDMA_SetChannelPreemptionConfig . . . . .	194
12.7.8	EDMA_SetChannelLink . . . . .	195
12.7.9	EDMA_SetBandWidth . . . . .	195
12.7.10	EDMA_SetModulo . . . . .	196
12.7.11	EDMA_EnableAutoStopRequest . . . . .	196
12.7.12	EDMA_EnableChannelInterrupts . . . . .	196
12.7.13	EDMA_DisableChannelInterrupts . . . . .	197

# Contents

Section Number	Title	Page Number
12.7.14	<a href="#">EDMA_TcdReset</a>	197
12.7.15	<a href="#">EDMA_TcdSetTransferConfig</a>	197
12.7.16	<a href="#">EDMA_TcdSetMinorOffsetConfig</a>	198
12.7.17	<a href="#">EDMA_TcdSetChannelLink</a>	198
12.7.18	<a href="#">EDMA_TcdSetBandWidth</a>	199
12.7.19	<a href="#">EDMA_TcdSetModulo</a>	199
12.7.20	<a href="#">EDMA_TcdEnableAutoStopRequest</a>	200
12.7.21	<a href="#">EDMA_TcdEnableInterrupts</a>	200
12.7.22	<a href="#">EDMA_TcdDisableInterrupts</a>	200
12.7.23	<a href="#">EDMA_EnableChannelRequest</a>	200
12.7.24	<a href="#">EDMA_DisableChannelRequest</a>	201
12.7.25	<a href="#">EDMA_TriggerChannelStart</a>	201
12.7.26	<a href="#">EDMA_GetRemainingBytes</a>	201
12.7.27	<a href="#">EDMA_GetErrorStatusFlags</a>	202
12.7.28	<a href="#">EDMA_GetChannelStatusFlags</a>	202
12.7.29	<a href="#">EDMA_ClearChannelStatusFlags</a>	202
12.7.30	<a href="#">EDMA_CreateHandle</a>	203
12.7.31	<a href="#">EDMA_InstallTCDMemory</a>	203
12.7.32	<a href="#">EDMA_SetCallback</a>	203
12.7.33	<a href="#">EDMA_PrepTransfer</a>	204
12.7.34	<a href="#">EDMA_SubmitTransfer</a>	204
12.7.35	<a href="#">EDMA_StartTransfer</a>	205
12.7.36	<a href="#">EDMA_StopTransfer</a>	205
12.7.37	<a href="#">EDMA_AbortTransfer</a>	205
12.7.38	<a href="#">EDMA_HandleIRQ</a>	206

## Chapter [EWM: External Watchdog Monitor Driver](#)

<b>13.1</b>	<a href="#">Overview</a>	<b>207</b>
<b>13.2</b>	<a href="#">Typical use case</a>	<b>207</b>
<b>13.3</b>	<a href="#">Data Structure Documentation</a>	<b>208</b>
13.3.1	<a href="#">struct ewm_config_t</a>	208
<b>13.4</b>	<a href="#">Macro Definition Documentation</a>	<b>208</b>
13.4.1	<a href="#">FSL_EWM_DRIVER_VERSION</a>	208
<b>13.5</b>	<a href="#">Enumeration Type Documentation</a>	<b>208</b>
13.5.1	<a href="#">_ewm_interrupt_enable_t</a>	208
13.5.2	<a href="#">_ewm_status_flags_t</a>	208
<b>13.6</b>	<a href="#">Function Documentation</a>	<b>209</b>
13.6.1	<a href="#">EWM_Init</a>	209
13.6.2	<a href="#">EWM_Deinit</a>	209

# Contents

Section Number	Title	Page Number
13.6.3	EWM_GetDefaultConfig . . . . .	209
13.6.4	EWM_EnableInterrupts . . . . .	210
13.6.5	EWM_DisableInterrupts . . . . .	210
13.6.6	EWM_GetStatusFlags . . . . .	210
13.6.7	EWM_Refresh . . . . .	211

## Chapter C90TFS Flash Driver

<b>14.1</b>	<b>Overview</b> . . . . .	<b>213</b>
<b>14.2</b>	<b>Data Structure Documentation</b> . . . . .	<b>221</b>
14.2.1	struct flash_execute_in_ram_function_config_t . . . . .	221
14.2.2	struct flash_swap_state_config_t . . . . .	221
14.2.3	struct flash_swap_ifr_field_config_t . . . . .	221
14.2.4	union flash_swap_ifr_field_data_t . . . . .	222
14.2.5	struct flash_operation_config_t . . . . .	222
14.2.6	struct flash_config_t . . . . .	223
<b>14.3</b>	<b>Macro Definition Documentation</b> . . . . .	<b>224</b>
14.3.1	MAKE_VERSION . . . . .	224
14.3.2	FSL_FLASH_DRIVER_VERSION . . . . .	224
14.3.3	FLASH_SSD_CONFIG_ENABLE_FLEXNVM_SUPPORT . . . . .	224
14.3.4	FLASH_DRIVER_IS_FLASH_RESIDENT . . . . .	224
14.3.5	FLASH_DRIVER_IS_EXPORTED . . . . .	224
14.3.6	kStatusGroupGeneric . . . . .	225
14.3.7	MAKE_STATUS . . . . .	225
14.3.8	FOUR_CHAR_CODE . . . . .	225
<b>14.4</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>225</b>
14.4.1	_flash_driver_version_constants . . . . .	225
14.4.2	_flash_status . . . . .	225
14.4.3	_flash_driver_api_keys . . . . .	226
14.4.4	flash_margin_value_t . . . . .	226
14.4.5	flash_security_state_t . . . . .	226
14.4.6	flash_protection_state_t . . . . .	226
14.4.7	flash_execute_only_access_state_t . . . . .	226
14.4.8	flash_property_tag_t . . . . .	227
14.4.9	_flash_execute_in_ram_function_constants . . . . .	227
14.4.10	flash_read_resource_option_t . . . . .	227
14.4.11	_flash_read_resource_range . . . . .	228
14.4.12	flash_flexram_function_option_t . . . . .	228
14.4.13	flash_swap_function_option_t . . . . .	228
14.4.14	flash_swap_control_option_t . . . . .	228
14.4.15	flash_swap_state_t . . . . .	229
14.4.16	flash_swap_block_status_t . . . . .	229

# Contents

Section Number	Title	Page Number
14.4.17	<a href="#">flash_partition_flexram_load_option_t</a>	229
<b>14.5</b>	<b>Function Documentation</b>	<b>229</b>
14.5.1	<a href="#">FLASH_Init</a>	229
14.5.2	<a href="#">FLASH_SetCallback</a>	230
14.5.3	<a href="#">FLASH_PrepareExecuteInRamFunctions</a>	230
14.5.4	<a href="#">FLASH_EraseAll</a>	231
14.5.5	<a href="#">FLASH_Erase</a>	232
14.5.6	<a href="#">FLASH_EraseAllExecuteOnlySegments</a>	233
14.5.7	<a href="#">FLASH_Program</a>	235
14.5.8	<a href="#">FLASH_ProgramOnce</a>	236
14.5.9	<a href="#">FLASH_ReadOnce</a>	237
14.5.10	<a href="#">FLASH_GetSecurityState</a>	239
14.5.11	<a href="#">FLASH_SecurityBypass</a>	240
14.5.12	<a href="#">FLASH_VerifyEraseAll</a>	241
14.5.13	<a href="#">FLASH_VerifyErase</a>	242
14.5.14	<a href="#">FLASH_VerifyProgram</a>	243
14.5.15	<a href="#">FLASH_VerifyEraseAllExecuteOnlySegments</a>	244
14.5.16	<a href="#">FLASH_IsProtected</a>	245
14.5.17	<a href="#">FLASH_IsExecuteOnly</a>	246
14.5.18	<a href="#">FLASHGetProperty</a>	246
14.5.19	<a href="#">FLASH_PflashSetProtection</a>	247
14.5.20	<a href="#">FLASH_PflashGetProtection</a>	247

## Chapter **FlexIO: FlexIO Driver**

<b>15.1</b>	<b>Overview</b>	<b>249</b>
<b>15.2</b>	<b>FlexIO Driver</b>	<b>250</b>
15.2.1	<a href="#">Overview</a>	250
15.2.2	<a href="#">Data Structure Documentation</a>	254
15.2.3	<a href="#">Macro Definition Documentation</a>	257
15.2.4	<a href="#">Typedef Documentation</a>	257
15.2.5	<a href="#">Enumeration Type Documentation</a>	257
15.2.6	<a href="#">Function Documentation</a>	261
<b>15.3</b>	<b>FlexIO Camera Driver</b>	<b>272</b>
15.3.1	<a href="#">Overview</a>	272
15.3.2	<a href="#">Typical use case</a>	272
15.3.3	<a href="#">Data Structure Documentation</a>	275
15.3.4	<a href="#">Macro Definition Documentation</a>	276
15.3.5	<a href="#">Enumeration Type Documentation</a>	276
15.3.6	<a href="#">Function Documentation</a>	277
15.3.7	<a href="#">FlexIO eDMA Camera Driver</a>	281

# Contents

Section Number	Title	Page Number
<b>15.4</b>	<b>FlexIO I2C Master Driver</b>	<b>285</b>
15.4.1	Overview . . . . .	285
15.4.2	Typical use case . . . . .	285
15.4.3	Data Structure Documentation . . . . .	289
15.4.4	Macro Definition Documentation . . . . .	292
15.4.5	Typedef Documentation . . . . .	292
15.4.6	Enumeration Type Documentation . . . . .	292
15.4.7	Function Documentation . . . . .	293
<b>15.5</b>	<b>FlexIO I2S Driver</b> . . . . .	<b>302</b>
15.5.1	Overview . . . . .	302
15.5.2	Typical use case . . . . .	302
15.5.3	Data Structure Documentation . . . . .	307
15.5.4	Macro Definition Documentation . . . . .	309
15.5.5	Enumeration Type Documentation . . . . .	309
15.5.6	Function Documentation . . . . .	311
15.5.7	FlexIO eDMA I2S Driver . . . . .	322
15.5.8	FlexIO DMA I2S Driver . . . . .	328
<b>15.6</b>	<b>FlexIO SPI Driver</b> . . . . .	<b>334</b>
15.6.1	Overview . . . . .	334
15.6.2	Typical use case . . . . .	334
15.6.3	Data Structure Documentation . . . . .	340
15.6.4	Macro Definition Documentation . . . . .	345
15.6.5	Typedef Documentation . . . . .	345
15.6.6	Enumeration Type Documentation . . . . .	345
15.6.7	Function Documentation . . . . .	347
15.6.8	FlexIO eDMA SPI Driver . . . . .	361
15.6.9	FlexIO DMA SPI Driver . . . . .	366
<b>15.7</b>	<b>FlexIO UART Driver</b> . . . . .	<b>371</b>
15.7.1	Overview . . . . .	371
15.7.2	Typical use case . . . . .	371
15.7.3	Data Structure Documentation . . . . .	379
15.7.4	Macro Definition Documentation . . . . .	382
15.7.5	Typedef Documentation . . . . .	382
15.7.6	Enumeration Type Documentation . . . . .	382
15.7.7	Function Documentation . . . . .	383
15.7.8	FlexIO eDMA UART Driver . . . . .	394
15.7.9	FlexIO DMA UART Driver . . . . .	399

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

<b>16.1</b>	<b>Overview</b> . . . . .	<b>405</b>
-------------	---------------------------	------------

# Contents

Section Number	Title	Page Number
<b>16.2</b>	<b>Data Structure Documentation</b>	<b>405</b>
16.2.1	struct gpio_pin_config_t . . . . .	405
<b>16.3</b>	<b>Macro Definition Documentation</b>	<b>406</b>
16.3.1	FSL_GPIO_DRIVER_VERSION . . . . .	406
<b>16.4</b>	<b>Enumeration Type Documentation</b>	<b>406</b>
16.4.1	gpio_pin_direction_t . . . . .	406
<b>16.5</b>	<b>GPIO Driver</b>	<b>407</b>
16.5.1	Overview . . . . .	407
16.5.2	Typical use case . . . . .	407
16.5.3	Function Documentation . . . . .	408
<b>16.6</b>	<b>FGPIO Driver</b>	<b>411</b>
16.6.1	Typical use case . . . . .	411

## Chapter I2C: Inter-Integrated Circuit Driver

<b>17.1</b>	<b>Overview</b>	<b>413</b>
<b>17.2</b>	<b>I2C Driver</b>	<b>414</b>
17.2.1	Overview . . . . .	414
17.2.2	Typical use case . . . . .	414
17.2.3	Data Structure Documentation . . . . .	421
17.2.4	Macro Definition Documentation . . . . .	425
17.2.5	Typedef Documentation . . . . .	425
17.2.6	Enumeration Type Documentation . . . . .	425
17.2.7	Function Documentation . . . . .	427
<b>17.3</b>	<b>I2C eDMA Driver</b>	<b>441</b>
17.3.1	Overview . . . . .	441
17.3.2	Data Structure Documentation . . . . .	441
17.3.3	Typedef Documentation . . . . .	442
17.3.4	Function Documentation . . . . .	442
<b>17.4</b>	<b>I2C DMA Driver</b>	<b>444</b>
17.4.1	Overview . . . . .	444
17.4.2	Data Structure Documentation . . . . .	444
17.4.3	Typedef Documentation . . . . .	445
17.4.4	Function Documentation . . . . .	445
<b>17.5</b>	<b>I2C FreeRTOS Driver</b>	<b>447</b>
17.5.1	Overview . . . . .	447
17.5.2	Data Structure Documentation . . . . .	447
17.5.3	Function Documentation . . . . .	448

# Contents

Section Number	Title	Page Number
<b>17.6</b>	<b>I2C µCOS/II Driver</b>	<b>450</b>
17.6.1	Overview	450
17.6.2	Data Structure Documentation	450
17.6.3	Function Documentation	451
<b>17.7</b>	<b>I2C µCOS/III Driver</b>	<b>453</b>
17.7.1	Overview	453
17.7.2	Data Structure Documentation	453
17.7.3	Function Documentation	454
<b>Chapter</b>	<b>INTMUX: Interrupt Multiplexer Driver</b>	
<b>18.1</b>	<b>Overview</b>	<b>457</b>
<b>18.2</b>	<b>Typical use case</b>	<b>457</b>
18.2.1	Channel Configure	457
<b>18.3</b>	<b>Macro Definition Documentation</b>	<b>458</b>
18.3.1	FSL_INTMUX_DRIVER_VERSION	458
<b>18.4</b>	<b>Enumeration Type Documentation</b>	<b>458</b>
18.4.1	intmux_channel_logic_mode_t	458
<b>18.5</b>	<b>Function Documentation</b>	<b>458</b>
18.5.1	INTMUX_Init	458
18.5.2	INTMUX_Deinit	458
18.5.3	INTMUX_ResetChannel	459
18.5.4	INTMUX_SetChannelMode	460
18.5.5	INTMUX_EnableInterrupt	460
18.5.6	INTMUX_DisableInterrupt	460
18.5.7	INTMUX_GetChannelPendingSources	461
<b>Chapter</b>	<b>LLWU: Low-Leakage Wakeup Unit Driver</b>	
<b>19.1</b>	<b>Overview</b>	<b>463</b>
<b>19.2</b>	<b>External wakeup pins configurations</b>	<b>463</b>
<b>19.3</b>	<b>Internal wakeup modules configurations</b>	<b>463</b>
<b>19.4</b>	<b>Digital pin filter for external wakeup pin configurations</b>	<b>463</b>
<b>19.5</b>	<b>Data Structure Documentation</b>	<b>464</b>
19.5.1	struct llwu_external_pin_filter_mode_t	464
<b>19.6</b>	<b>Macro Definition Documentation</b>	<b>464</b>

# Contents

Section Number	Title	Page Number
19.6.1	FSL_LLWU_DRIVER_VERSION . . . . .	464
19.7	<b>Enumeration Type Documentation</b> . . . . .	464
19.7.1	llwu_external_pin_mode_t . . . . .	464
19.7.2	llwu_pin_filter_mode_t . . . . .	465
19.8	<b>Function Documentation</b> . . . . .	465
19.8.1	LLWU_SetExternalWakeUpPinMode . . . . .	465
19.8.2	LLWU_GetExternalWakeUpPinFlag . . . . .	465
19.8.3	LLWU_ClearExternalWakeUpPinFlag . . . . .	466
19.8.4	LLWU_EnableInternalModuleInterruptWakup . . . . .	467
19.8.5	LLWU_GetInternalWakeUpModuleFlag . . . . .	467
19.8.6	LLWU_SetPinFilterMode . . . . .	467
19.8.7	LLWU_GetPinFilterFlag . . . . .	468
19.8.8	LLWU_ClearPinFilterFlag . . . . .	468
<b>Chapter LPTMR: Low-Power Timer</b>		
20.1	<b>Overview</b> . . . . .	469
20.2	<b>Function groups</b> . . . . .	469
20.2.1	Initialization and deinitialization . . . . .	469
20.2.2	Timer period Operations . . . . .	469
20.2.3	Start and Stop timer operations . . . . .	469
20.2.4	Status . . . . .	470
20.2.5	Interrupt . . . . .	470
20.3	<b>Typical use case</b> . . . . .	470
20.3.1	LPTMR tick example . . . . .	470
20.4	<b>Data Structure Documentation</b> . . . . .	472
20.4.1	struct lptmr_config_t . . . . .	472
20.5	<b>Enumeration Type Documentation</b> . . . . .	473
20.5.1	lptmr_pin_select_t . . . . .	473
20.5.2	lptmr_pin_polarity_t . . . . .	473
20.5.3	lptmr_timer_mode_t . . . . .	473
20.5.4	lptmr_prescaler_glitch_value_t . . . . .	474
20.5.5	lptmr_prescaler_clock_select_t . . . . .	474
20.5.6	lptmr_interrupt_enable_t . . . . .	474
20.5.7	lptmr_status_flags_t . . . . .	475
20.6	<b>Function Documentation</b> . . . . .	475
20.6.1	LPTMR_Init . . . . .	475
20.6.2	LPTMR_Deinit . . . . .	475
20.6.3	LPTMR_GetDefaultConfig . . . . .	475

# Contents

Section Number	Title	Page Number
20.6.4	LPTMR_EnableInterrupts . . . . .	476
20.6.5	LPTMR_DisableInterrupts . . . . .	476
20.6.6	LPTMR_GetEnabledInterrupts . . . . .	476
20.6.7	LPTMR_GetStatusFlags . . . . .	476
20.6.8	LPTMR_ClearStatusFlags . . . . .	477
20.6.9	LPTMR_SetTimerPeriod . . . . .	477
20.6.10	LPTMR_GetCurrentTimerCount . . . . .	477
20.6.11	LPTMR_StartTimer . . . . .	478
20.6.12	LPTMR_StopTimer . . . . .	478

## Chapter LPUART: Low Power UART Driver

<b>21.1</b>	<b>Overview</b> . . . . .	<b>479</b>
<b>21.2</b>	<b>LPUART Driver</b> . . . . .	<b>480</b>
21.2.1	Overview . . . . .	480
21.2.2	Typical use case . . . . .	480
21.2.3	Data Structure Documentation . . . . .	484
21.2.4	Macro Definition Documentation . . . . .	486
21.2.5	Typedef Documentation . . . . .	486
21.2.6	Enumeration Type Documentation . . . . .	486
21.2.7	Function Documentation . . . . .	488
<b>21.3</b>	<b>LPUART DMA Driver</b> . . . . .	<b>500</b>
21.3.1	Overview . . . . .	500
21.3.2	Data Structure Documentation . . . . .	500
21.3.3	Typedef Documentation . . . . .	501
21.3.4	Function Documentation . . . . .	501
<b>21.4</b>	<b>LPUART eDMA Driver</b> . . . . .	<b>505</b>
21.4.1	Overview . . . . .	505
21.4.2	Data Structure Documentation . . . . .	506
21.4.3	Typedef Documentation . . . . .	507
21.4.4	Function Documentation . . . . .	507
<b>21.5</b>	<b>LPUART μCOS/II Driver</b> . . . . .	<b>511</b>
21.5.1	Overview . . . . .	511
21.5.2	Function Documentation . . . . .	511
<b>21.6</b>	<b>LPUART μCOS/III Driver</b> . . . . .	<b>513</b>
21.6.1	Overview . . . . .	513
21.6.2	Function Documentation . . . . .	513
<b>21.7</b>	<b>LPUART FreeRTOS Driver</b> . . . . .	<b>515</b>
21.7.1	Overview . . . . .	515
21.7.2	Function Documentation . . . . .	515

# Contents

Section Number	Title	Page Number
Chapter	<b>LTC: LP Trusted Cryptography</b>	
22.1	<b>Overview</b> . . . . .	517
22.2	<b>LTC Driver Initialization and Configuration</b> . . . . .	517
22.3	<b>Comments about API usage in RTOS</b> . . . . .	517
22.4	<b>Comments about API usage in interrupt handler</b> . . . . .	517
22.5	<b>LTC Driver Examples</b> . . . . .	518
22.5.1	Simple examples . . . . .	518
22.6	<b>Macro Definition Documentation</b> . . . . .	520
22.6.1	FSL_LTC_DRIVER_VERSION . . . . .	520
22.7	<b>Function Documentation</b> . . . . .	520
22.7.1	LTC_Init . . . . .	520
22.7.2	LTC_Deinit . . . . .	520
22.7.3	LTC_SetDpaMaskSeed . . . . .	521
22.8	<b>LTC Blocking APIs</b> . . . . .	522
22.8.1	Overview . . . . .	522
22.8.2	LTC DES driver . . . . .	523
22.8.3	LTC AES driver . . . . .	540
22.8.4	LTC HASH driver . . . . .	550
22.8.5	LTC PKHA driver . . . . .	555
22.9	<b>LTC Non-blocking eDMA APIs</b> . . . . .	571
22.9.1	Overview . . . . .	571
22.9.2	Data Structure Documentation . . . . .	571
22.9.3	Typedef Documentation . . . . .	573
22.9.4	Function Documentation . . . . .	574
22.9.5	LTC eDMA DES driver . . . . .	576
22.9.6	LTC eDMA AES driver . . . . .	596
Chapter	<b>MPU: Memory Protection Unit</b>	
23.1	<b>Overview</b> . . . . .	603
23.2	<b>Initialization and Deinitialize</b> . . . . .	603
23.3	<b>Basic Control Operations</b> . . . . .	604
23.4	<b>Data Structure Documentation</b> . . . . .	607
23.4.1	struct mpu.hardware_info_t . . . . .	607
23.4.2	struct mpu.access_err_info_t . . . . .	607

# Contents

Section Number	Title	Page Number
23.4.3	struct mpu_rwxrights_master_access_control_t . . . . .	608
23.4.4	struct mpu_rwrights_master_access_control_t . . . . .	608
23.4.5	struct mpu_region_config_t . . . . .	608
23.4.6	struct mpu_config_t . . . . .	609
<b>23.5</b>	<b>Macro Definition Documentation</b> . . . . .	<b>611</b>
23.5.1	FSL MPU_DRIVER_VERSION . . . . .	611
23.5.2	MPU_REGION_RWXRIGHTS_MASTER_SHIFT . . . . .	611
23.5.3	MPU_REGION_RWXRIGHTS_MASTER_MASK . . . . .	611
23.5.4	MPU_REGION_RWXRIGHTS_MASTER_WIDTH . . . . .	611
23.5.5	MPU_REGION_RWXRIGHTS_MASTER . . . . .	611
23.5.6	MPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT . . . . .	611
23.5.7	MPU_REGION_RWXRIGHTS_MASTER_PE_MASK . . . . .	611
23.5.8	MPU_REGION_RWXRIGHTS_MASTER_PE . . . . .	611
23.5.9	MPU_REGION_RWRIGHTS_MASTER_SHIFT . . . . .	611
23.5.10	MPU_REGION_RWRIGHTS_MASTER_MASK . . . . .	611
23.5.11	MPU_REGION_RWRIGHTS_MASTER . . . . .	611
<b>23.6</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>612</b>
23.6.1	mpu_region_total_num_t . . . . .	612
23.6.2	mpu_slave_t . . . . .	612
23.6.3	mpu_err_access_control_t . . . . .	612
23.6.4	mpu_err_access_type_t . . . . .	612
23.6.5	mpu_err_attributes_t . . . . .	612
23.6.6	mpu_supervisor_access_rights_t . . . . .	613
23.6.7	mpu_user_access_rights_t . . . . .	613
<b>23.7</b>	<b>Function Documentation</b> . . . . .	<b>613</b>
23.7.1	MPU_Init . . . . .	613
23.7.2	MPU_Deinit . . . . .	613
23.7.3	MPU_Enable . . . . .	614
23.7.4	MPU_RegionEnable . . . . .	614
23.7.5	MPU_GetHardwareInfo . . . . .	614
23.7.6	MPU_SetRegionConfig . . . . .	615
23.7.7	MPU_SetRegionAddr . . . . .	615
23.7.8	MPU_SetRegionRwxMasterAccessRights . . . . .	615
23.7.9	MPU_SetRegionRwMasterAccessRights . . . . .	616
23.7.10	MPU_GetSlavePortErrorStatus . . . . .	616
23.7.11	MPU_GetDetailErrorAccessInfo . . . . .	617
 <b>Chapter PIT: Periodic Interrupt Timer</b>		
<b>24.1</b>	<b>Overview</b> . . . . .	<b>619</b>
<b>24.2</b>	<b>Function groups</b> . . . . .	<b>619</b>

# Contents

Section Number	Title	Page Number
24.2.1	Initialization and deinitialization . . . . .	619
24.2.2	Timer period Operations . . . . .	619
24.2.3	Start and Stop timer operations . . . . .	619
24.2.4	Status . . . . .	620
24.2.5	Interrupt . . . . .	620
<b>24.3</b>	<b>Typical use case . . . . .</b>	<b>620</b>
24.3.1	PIT tick example . . . . .	620
<b>24.4</b>	<b>Data Structure Documentation . . . . .</b>	<b>622</b>
24.4.1	struct pit_config_t . . . . .	622
<b>24.5</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>622</b>
24.5.1	pit_chnl_t . . . . .	622
24.5.2	pit_interrupt_enable_t . . . . .	623
24.5.3	pit_status_flags_t . . . . .	623
<b>24.6</b>	<b>Function Documentation . . . . .</b>	<b>623</b>
24.6.1	PIT_Init . . . . .	623
24.6.2	PIT_Deinit . . . . .	623
24.6.3	PIT_GetDefaultConfig . . . . .	623
24.6.4	PIT_EnableInterrupts . . . . .	624
24.6.5	PIT_DisableInterrupts . . . . .	624
24.6.6	PIT_GetEnabledInterrupts . . . . .	624
24.6.7	PIT_GetStatusFlags . . . . .	625
24.6.8	PIT_ClearStatusFlags . . . . .	626
24.6.9	PIT_SetTimerPeriod . . . . .	626
24.6.10	PIT_GetCurrentTimerCount . . . . .	627
24.6.11	PIT_StartTimer . . . . .	627
24.6.12	PIT_StopTimer . . . . .	627

## Chapter **PMC: Power Management Controller**

<b>25.1</b>	<b>Overview . . . . .</b>	<b>629</b>
<b>25.2</b>	<b>Data Structure Documentation . . . . .</b>	<b>629</b>
25.2.1	struct pmc_low_volt_detect_config_t . . . . .	629
25.2.2	struct pmc_low_volt_warning_config_t . . . . .	630
<b>25.3</b>	<b>Macro Definition Documentation . . . . .</b>	<b>630</b>
25.3.1	FSL_PMC_DRIVER_VERSION . . . . .	630
<b>25.4</b>	<b>Function Documentation . . . . .</b>	<b>630</b>
25.4.1	PMC_ConfigureLowVoltDetect . . . . .	630
25.4.2	PMC_GetLowVoltDetectFlag . . . . .	630
25.4.3	PMC_ClearLowVoltDetectFlag . . . . .	631

# Contents

Section Number	Title	Page Number
25.4.4	PMC_ConfigureLowVoltWarning . . . . .	631
25.4.5	PMC_GetLowVoltWarningFlag . . . . .	631
25.4.6	PMC_ClearLowVoltWarningFlag . . . . .	632

## Chapter PORT: Port Control and Interrupts

26.1	<b>Overview</b> . . . . .	635
26.2	<b>Typical configuration use case</b> . . . . .	635
26.2.1	Input PORT configuration . . . . .	635
26.2.2	I2C PORT Configuration . . . . .	635
26.3	<b>Data Structure Documentation</b> . . . . .	637
26.3.1	struct port_pin_config_t . . . . .	637
26.4	<b>Macro Definition Documentation</b> . . . . .	637
26.4.1	FSL_PORT_DRIVER_VERSION . . . . .	637
26.5	<b>Enumeration Type Documentation</b> . . . . .	637
26.5.1	_port_pull . . . . .	637
26.5.2	_port_slew_rate . . . . .	637
26.5.3	_port_passive_filter_enable . . . . .	638
26.5.4	_port_drive_strength . . . . .	638
26.5.5	port_mux_t . . . . .	638
26.5.6	port_interrupt_t . . . . .	638
26.6	<b>Function Documentation</b> . . . . .	638
26.6.1	PORT_SetPinConfig . . . . .	638
26.6.2	PORT_SetMultiplePinsConfig . . . . .	639
26.6.3	PORT_SetPinMux . . . . .	639
26.6.4	PORT_SetPinInterruptConfig . . . . .	641
26.6.5	PORT_GetPinsInterruptFlags . . . . .	641
26.6.6	PORT_ClearPinsInterruptFlags . . . . .	642

## Chapter QSPI: Quad Serial Peripheral Interface Driver

27.1	<b>Overview</b> . . . . .	643
27.2	<b>Data Structure Documentation</b> . . . . .	647
27.2.1	struct qspi_dqs_config_t . . . . .	647
27.2.2	struct qspi_flash_timing_t . . . . .	648
27.2.3	struct qspi_config_t . . . . .	648
27.2.4	struct qspi_flash_config_t . . . . .	649
27.2.5	struct qspi_transfer_t . . . . .	650
27.3	<b>Macro Definition Documentation</b> . . . . .	650

# Contents

Section Number	Title	Page Number
27.3.1	<a href="#">FSL_QSPI_DRIVER_VERSION</a>	650
<b>27.4</b>	<b><a href="#">Enumeration Type Documentation</a></b>	<b>650</b>
27.4.1	<a href="#">_status_t</a>	650
27.4.2	<a href="#">qspi_read_area_t</a>	650
27.4.3	<a href="#">qspi_command_seq_t</a>	650
27.4.4	<a href="#">qspi_fifo_t</a>	651
27.4.5	<a href="#">qspi_endianness_t</a>	651
27.4.6	<a href="#">_qspi_error_flags</a>	651
27.4.7	<a href="#">_qspi_flags</a>	651
27.4.8	<a href="#">_qspi_interrupt_enable</a>	652
27.4.9	<a href="#">_qspi_dma_enable</a>	653
27.4.10	<a href="#">qspi_dqs_phrase_shift_t</a>	653
<b>27.5</b>	<b><a href="#">Function Documentation</a></b>	<b>653</b>
27.5.1	<a href="#">QSPI_Init</a>	653
27.5.2	<a href="#">QSPI_GetDefaultQspiConfig</a>	653
27.5.3	<a href="#">QSPI_Deinit</a>	654
27.5.4	<a href="#">QSPI_SetFlashConfig</a>	654
27.5.5	<a href="#">QSPI_SoftwareReset</a>	654
27.5.6	<a href="#">QSPI_Enable</a>	654
27.5.7	<a href="#">QSPI_GetStatusFlags</a>	655
27.5.8	<a href="#">QSPI_GetErrorStatusFlags</a>	655
27.5.9	<a href="#">QSPI_ClearErrorFlag</a>	655
27.5.10	<a href="#">QSPI_EnableInterrupts</a>	656
27.5.11	<a href="#">QSPI_DisableInterrupts</a>	656
27.5.12	<a href="#">QSPI_EnableDMA</a>	656
27.5.13	<a href="#">QSPI_GetTxDataRegisterAddress</a>	656
27.5.14	<a href="#">QSPI_GetRxDataRegisterAddress</a>	657
27.5.15	<a href="#">QSPI_SetIPCommandAddress</a>	657
27.5.16	<a href="#">QSPI_SetIPCommandSize</a>	657
27.5.17	<a href="#">QSPI_ExecuteIPCommand</a>	658
27.5.18	<a href="#">QSPI_ExecuteAHBCommand</a>	658
27.5.19	<a href="#">QSPI_EnableIPPParallelMode</a>	658
27.5.20	<a href="#">QSPI_EnableAHBParallelMode</a>	658
27.5.21	<a href="#">QSPI_UpdateLUT</a>	659
27.5.22	<a href="#">QSPI_ClearFifo</a>	659
27.5.23	<a href="#">QSPI_ClearCommandSequence</a>	659
27.5.24	<a href="#">QSPI_SetReadDataArea</a>	659
27.5.25	<a href="#">QSPI_WriteBlocking</a>	660
27.5.26	<a href="#">QSPI_WriteData</a>	660
27.5.27	<a href="#">QSPI_ReadBlocking</a>	660
27.5.28	<a href="#">QSPI_ReadData</a>	661
27.5.29	<a href="#">QSPI_TransferSendBlocking</a>	661
27.5.30	<a href="#">QSPI_TransferReceiveBlocking</a>	661

# Contents

Section Number	Title	Page Number
<b>27.6</b>	<b>QSPI eDMA Driver</b>	<b>663</b>
27.6.1	Overview . . . . .	663
27.6.2	Data Structure Documentation . . . . .	664
27.6.3	Function Documentation . . . . .	664
<b>Chapter RCM: Reset Control Module Driver</b>		
<b>28.1</b>	<b>Overview</b> . . . . .	<b>669</b>
<b>28.2</b>	<b>Data Structure Documentation</b> . . . . .	<b>670</b>
28.2.1	struct rcm_reset_pin_filter_config_t . . . . .	670
<b>28.3</b>	<b>Macro Definition Documentation</b> . . . . .	<b>670</b>
28.3.1	FSL_RCM_DRIVER_VERSION . . . . .	670
<b>28.4</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>670</b>
28.4.1	rcm_reset_source_t . . . . .	670
28.4.2	rcm_run_wait_filter_mode_t . . . . .	670
<b>28.5</b>	<b>Function Documentation</b> . . . . .	<b>671</b>
28.5.1	RCM_GetPreviousResetSources . . . . .	671
28.5.2	RCM_ConfigureResetPinFilter . . . . .	671
<b>Chapter RTC: Real Time Clock</b>		
<b>29.1</b>	<b>Overview</b> . . . . .	<b>673</b>
<b>29.2</b>	<b>Function groups</b> . . . . .	<b>673</b>
29.2.1	Initialization and deinitialization . . . . .	673
29.2.2	Set & Get Datetime . . . . .	673
29.2.3	Set & Get Alarm . . . . .	673
29.2.4	Start & Stop timer . . . . .	674
29.2.5	Status . . . . .	674
29.2.6	Interrupt . . . . .	674
29.2.7	RTC Oscillator . . . . .	674
29.2.8	Monotonic Counter . . . . .	674
<b>29.3</b>	<b>Typical use case</b> . . . . .	<b>674</b>
29.3.1	RTC tick example . . . . .	674
<b>29.4</b>	<b>Data Structure Documentation</b> . . . . .	<b>677</b>
29.4.1	struct rtc_datetime_t . . . . .	677
29.4.2	struct rtc_config_t . . . . .	678
<b>29.5</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>678</b>
29.5.1	rtc_interrupt_enable_t . . . . .	678

# Contents

Section Number	Title	Page Number
29.5.2	rtc_status_flags_t . . . . .	679
<b>29.6</b>	<b>Function Documentation</b> . . . . .	<b>679</b>
29.6.1	RTC_Init . . . . .	679
29.6.2	RTC_Deinit . . . . .	679
29.6.3	RTC_GetDefaultConfig . . . . .	679
29.6.4	RTC_SetDatetime . . . . .	680
29.6.5	RTC_GetDatetime . . . . .	680
29.6.6	RTC_SetAlarm . . . . .	680
29.6.7	RTC_GetAlarm . . . . .	681
29.6.8	RTC_EnableInterrupts . . . . .	681
29.6.9	RTC_DisableInterrupts . . . . .	681
29.6.10	RTC_GetEnabledInterrupts . . . . .	681
29.6.11	RTC_GetStatusFlags . . . . .	682
29.6.12	RTC_ClearStatusFlags . . . . .	682
29.6.13	RTC_StartTimer . . . . .	682
29.6.14	RTC_StopTimer . . . . .	682
29.6.15	RTC_Reset . . . . .	683

## Chapter SIM: System Integration Module Driver

<b>30.1</b>	<b>Overview</b> . . . . .	<b>685</b>
<b>30.2</b>	<b>Data Structure Documentation</b> . . . . .	<b>685</b>
30.2.1	struct sim_uid_t . . . . .	685
<b>30.3</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>686</b>
30.3.1	_sim_flash_mode . . . . .	686
<b>30.4</b>	<b>Function Documentation</b> . . . . .	<b>686</b>
30.4.1	SIM_GetUniqueId . . . . .	686
30.4.2	SIM_SetFlashMode . . . . .	686

## Chapter Smart Card

<b>31.1</b>	<b>Overview</b> . . . . .	<b>687</b>
<b>31.2</b>	<b>SmartCard Driver Initialization</b> . . . . .	<b>687</b>
<b>31.3</b>	<b>SmartCard Call diagram</b> . . . . .	<b>687</b>
<b>31.4</b>	<b>PHY driver</b> . . . . .	<b>687</b>
<b>31.5</b>	<b>Data Structure Documentation</b> . . . . .	<b>689</b>
31.5.1	struct smartcard_card_params_t . . . . .	689
31.5.2	struct smartcard_timers_state_t . . . . .	690

# Contents

Section Number	Title	Page Number
31.5.3	struct smartcard_interface_config_t . . . . .	691
31.5.4	struct smartcard_xfer_t . . . . .	692
31.5.5	struct smartcard_context_t . . . . .	692
<b>31.6</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>694</b>
31.6.1	smartcard_status_t . . . . .	694
31.6.2	smartcard_control_t . . . . .	694
31.6.3	smartcard_direction_t . . . . .	694
<b>31.7</b>	<b>Smart Card PHY TDA8035 Driver</b> . . . . .	<b>695</b>
31.7.1	Overview . . . . .	695
31.7.2	Macro Definition Documentation . . . . .	696
31.7.3	Function Documentation . . . . .	696
<b>Chapter</b>	<b>Smart Card PHY TDA8035 Driver</b>	
<b>32.1</b>	<b>Overview</b> . . . . .	<b>699</b>
<b>32.2</b>	<b>Macro Definition Documentation</b> . . . . .	<b>700</b>
32.2.1	SMARTCARD_NCN8025_STATUS_PRES . . . . .	700
<b>32.3</b>	<b>Function Documentation</b> . . . . .	<b>700</b>
32.3.1	SMARTCARD_PHY_NCN8025_GetDefaultConfig . . . . .	700
32.3.2	SMARTCARD_PHY_NCN8025_Init . . . . .	700
32.3.3	SMARTCARD_PHY_NCN8025_Deinit . . . . .	700
32.3.4	SMARTCARD_PHY_NCN8025_Activate . . . . .	701
32.3.5	SMARTCARD_PHY_NCN8025_Deactivate . . . . .	701
32.3.6	SMARTCARD_PHY_NCN8025_Control . . . . .	701
32.3.7	SMARTCARD_PHY_NCN8025_IRQHandler . . . . .	702
<b>32.4</b>	<b>Smart Card PHY EMVSIM Driver</b> . . . . .	<b>703</b>
32.4.1	Overview . . . . .	703
32.4.2	Function Documentation . . . . .	703
<b>32.5</b>	<b>Smart Card PHY GPIO Driver</b> . . . . .	<b>707</b>
32.5.1	Overview . . . . .	707
32.5.2	Function Documentation . . . . .	707
<b>32.6</b>	<b>Smart Card UART Driver</b> . . . . .	<b>711</b>
32.6.1	Overview . . . . .	711
32.6.2	Function Documentation . . . . .	712
<b>32.7</b>	<b>Smart Card EMVSIM Driver</b> . . . . .	<b>717</b>
32.7.1	Overview . . . . .	717
32.7.2	Enumeration Type Documentation . . . . .	718
32.7.3	Function Documentation . . . . .	719

# Contents

Section Number	Title	Page Number
<b>32.8</b>	<b>Smart Card FreeRTOS Driver</b>	<b>723</b>
32.8.1	Overview . . . . .	723
32.8.2	Data Structure Documentation . . . . .	724
32.8.3	Macro Definition Documentation . . . . .	725
32.8.4	Function Documentation . . . . .	725
<b>32.9</b>	<b>Smart Card μCOS/II Driver</b>	<b>729</b>
32.9.1	Overview . . . . .	729
32.9.2	Data Structure Documentation . . . . .	730
32.9.3	Macro Definition Documentation . . . . .	731
32.9.4	Function Documentation . . . . .	731
<b>32.10</b>	<b>Smart Card μCOS/III Driver</b>	<b>735</b>
32.10.1	Overview . . . . .	735
32.10.2	Data Structure Documentation . . . . .	736
32.10.3	Macro Definition Documentation . . . . .	737
32.10.4	Function Documentation . . . . .	737
 <b>Chapter SMC: System Mode Controller Driver</b>		
<b>33.1</b>	<b>Overview</b> . . . . .	<b>741</b>
<b>33.2</b>	<b>Macro Definition Documentation</b> . . . . .	<b>742</b>
33.2.1	FSL_SMC_DRIVER_VERSION . . . . .	742
<b>33.3</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>742</b>
33.3.1	smc_power_mode_protection_t . . . . .	742
33.3.2	smc_power_state_t . . . . .	742
33.3.3	smc_run_mode_t . . . . .	743
33.3.4	smc_stop_mode_t . . . . .	743
33.3.5	smc_partial_stop_option_t . . . . .	743
33.3.6	_smcmc_status . . . . .	743
<b>33.4</b>	<b>Function Documentation</b> . . . . .	<b>743</b>
33.4.1	SMC_SetPowerModeProtection . . . . .	743
33.4.2	SMC_GetPowerModeState . . . . .	744
33.4.3	SMC_SetPowerModeRun . . . . .	744
33.4.4	SMC_SetPowerModeWait . . . . .	744
33.4.5	SMC_SetPowerModeStop . . . . .	745
33.4.6	SMC_SetPowerModeVlpr . . . . .	745
33.4.7	SMC_SetPowerModeVlpw . . . . .	745
33.4.8	SMC_SetPowerModeVlps . . . . .	746

# Contents

Section Number	Title	Page Number
<b>Chapter</b>	<b>TPM: Timer PWM Module</b>	
<b>34.1</b>	<b>Overview</b>	<b>747</b>
<b>34.2</b>	<b>Typical use case</b>	<b>748</b>
34.2.1	PWM output	748
<b>34.3</b>	<b>Data Structure Documentation</b>	<b>752</b>
34.3.1	struct tpm_chnl_pwm_signal_param_t	752
34.3.2	struct tpm_config_t	752
<b>34.4</b>	<b>Enumeration Type Documentation</b>	<b>753</b>
34.4.1	tpm_chnl_t	753
34.4.2	tpm_pwm_mode_t	753
34.4.3	tpm_pwm_level_select_t	753
34.4.4	tpm_trigger_select_t	754
34.4.5	tpm_output_compare_mode_t	754
34.4.6	tpm_input_capture_edge_t	754
34.4.7	tpm_clock_source_t	754
34.4.8	tpm_clock_prescale_t	754
34.4.9	tpm_interrupt_enable_t	755
34.4.10	tpm_status_flags_t	755
<b>34.5</b>	<b>Function Documentation</b>	<b>755</b>
34.5.1	TPM_Init	755
34.5.2	TPM_Deinit	756
34.5.3	TPM_GetDefaultConfig	756
34.5.4	TPM_SetupPwm	756
34.5.5	TPM_UpdatePwmDutycycle	757
34.5.6	TPM_UpdateChnlEdgeLevelSelect	757
34.5.7	TPM_SetupInputCapture	758
34.5.8	TPM_SetupOutputCompare	758
34.5.9	TPM_EnableInterrupts	758
34.5.10	TPM_DisableInterrupts	759
34.5.11	TPM_GetEnabledInterrupts	759
34.5.12	TPM_GetStatusFlags	759
34.5.13	TPM_ClearStatusFlags	759
34.5.14	TPM_StartTimer	760
34.5.15	TPM_StopTimer	760
<b>Chapter</b>	<b>TRNG: True Random Number Generator</b>	
<b>35.1</b>	<b>Overview</b>	<b>761</b>
<b>35.2</b>	<b>TRNG Initialization</b>	<b>761</b>

# Contents

Section Number	Title	Page Number
35.3	Get random data from TRNG . . . . .	761
35.4	Data Structure Documentation . . . . .	763
35.4.1	struct trng_statistical_check_limit_t . . . . .	763
35.4.2	struct trng_config_t . . . . .	763
35.5	Macro Definition Documentation . . . . .	765
35.5.1	FSL_TRNG_DRIVER_VERSION . . . . .	765
35.6	Enumeration Type Documentation . . . . .	765
35.6.1	trng_sample_mode_t . . . . .	765
35.6.2	trng_clock_mode_t . . . . .	766
35.6.3	trng_ring_osc_div_t . . . . .	766
35.7	Function Documentation . . . . .	766
35.7.1	TRNG_GetDefaultConfig . . . . .	766
35.7.2	TRNG_Init . . . . .	767
35.7.3	TRNG_Deinit . . . . .	767
35.7.4	TRNG_GetRandomData . . . . .	767
<b>Chapter VREF: Voltage Reference Driver</b>		
36.1	Overview . . . . .	769
36.2	Typical use case and example . . . . .	769
36.3	Data Structure Documentation . . . . .	770
36.3.1	struct vref_config_t . . . . .	770
36.4	Macro Definition Documentation . . . . .	770
36.4.1	FSL_VREF_DRIVER_VERSION . . . . .	770
36.5	Enumeration Type Documentation . . . . .	770
36.5.1	vref_buffer_mode_t . . . . .	770
36.6	Function Documentation . . . . .	770
36.6.1	VREF_Init . . . . .	770
36.6.2	VREF_Deinit . . . . .	771
36.6.3	VREF_GetDefaultConfig . . . . .	771
36.6.4	VREF_SetTrimVal . . . . .	771
36.6.5	VREF_GetTrimVal . . . . .	772
<b>Chapter WDOG: Watchdog Timer Driver</b>		
37.1	Overview . . . . .	773

## Contents

Section Number		Page Number
	Title	
<b>37.2</b>	<b>Typical use case</b>	<b>773</b>
<b>37.3</b>	<b>Data Structure Documentation</b>	<b>775</b>
37.3.1	struct wdog_work_mode_t	775
37.3.2	struct wdog_config_t	775
37.3.3	struct wdog_test_config_t	776
<b>37.4</b>	<b>Macro Definition Documentation</b>	<b>776</b>
37.4.1	FSL_WDOG_DRIVER_VERSION	776
<b>37.5</b>	<b>Enumeration Type Documentation</b>	<b>776</b>
37.5.1	wdog_clock_source_t	776
37.5.2	wdog_clock_prescaler_t	776
37.5.3	wdog_test_mode_t	777
37.5.4	wdog_tested_byte_t	777
37.5.5	_wdog_interrupt_enable_t	777
37.5.6	_wdog_status_flags_t	777
<b>37.6</b>	<b>Function Documentation</b>	<b>777</b>
37.6.1	WDOG_GetDefaultConfig	777
37.6.2	WDOG_Init	778
37.6.3	WDOG_Deinit	778
37.6.4	WDOG_SetTestModeConfig	778
37.6.5	WDOG_Enable	779
37.6.6	WDOG_Disable	779
37.6.7	WDOG_EnableInterrupts	779
37.6.8	WDOG_DisableInterrupts	780
37.6.9	WDOG_GetStatusFlags	780
37.6.10	WDOG_ClearStatusFlags	781
37.6.11	WDOG_SetTimeoutValue	781
37.6.12	WDOG_SetWindowValue	781
37.6.13	WDOG_Unlock	782
37.6.14	WDOG_Refresh	782
37.6.15	WDOG_GetResetCount	782
37.6.16	WDOG_ClearResetCount	783

## Chapter **Debug Console**

<b>38.1</b>	<b>Overview</b>	<b>785</b>
<b>38.2</b>	<b>Function groups</b>	<b>785</b>
38.2.1	Initialization	785
38.2.2	Advanced Feature	786
<b>38.3</b>	<b>Typical use case</b>	<b>789</b>

# Contents

Section Number	Title	Page Number
<b>38.4</b>	<b>Semihosting</b>	<b>791</b>
38.4.1	Guide Semihosting for IAR . . . . .	791
38.4.2	Guide Semihosting for Keil µVision . . . . .	791
38.4.3	Guide Semihosting for KDS . . . . .	793
38.4.4	Guide Semihosting for ATL . . . . .	793
38.4.5	Guide Semihosting for ARMGCC . . . . .	794

## Chapter Notification Framework

<b>39.1</b>	<b>Overview</b> . . . . .	<b>797</b>
<b>39.2</b>	<b>Notifier Overview</b> . . . . .	<b>797</b>
<b>39.3</b>	<b>Data Structure Documentation</b> . . . . .	<b>799</b>
39.3.1	struct notifier_notification_block_t . . . . .	799
39.3.2	struct notifier_callback_config_t . . . . .	800
39.3.3	struct notifier_handle_t . . . . .	800
<b>39.4</b>	<b>Typedef Documentation</b> . . . . .	<b>801</b>
39.4.1	notifier_user_config_t . . . . .	801
39.4.2	notifier_user_function_t . . . . .	801
39.4.3	notifier_callback_t . . . . .	802
<b>39.5</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>802</b>
39.5.1	_notifier_status . . . . .	802
39.5.2	notifier_policy_t . . . . .	803
39.5.3	notifier_notification_type_t . . . . .	803
39.5.4	notifier_callback_type_t . . . . .	803
<b>39.6</b>	<b>Function Documentation</b> . . . . .	<b>804</b>
39.6.1	NOTIFIER_CreateHandle . . . . .	804
39.6.2	NOTIFIER_SwitchConfig . . . . .	805
39.6.3	NOTIFIER_GetErrorCallbackIndex . . . . .	806

## Chapter Shell

<b>40.1</b>	<b>Overview</b> . . . . .	<b>807</b>
<b>40.2</b>	<b>Function groups</b> . . . . .	<b>807</b>
40.2.1	Initialization . . . . .	807
40.2.2	Advanced Feature . . . . .	807
40.2.3	Shell Operation . . . . .	808
<b>40.3</b>	<b>Data Structure Documentation</b> . . . . .	<b>809</b>
40.3.1	struct shell_context_struct . . . . .	809
40.3.2	struct shell_command_context_t . . . . .	810

## Contents

Section Number	Title	Page Number
40.3.3	struct shell_command_context_list_t . . . . .	810
<b>40.4</b>	<b>Macro Definition Documentation</b> . . . . .	<b>811</b>
40.4.1	SHELL_USE_HISTORY . . . . .	811
40.4.2	SHELL_SEARCH_IN_HIST . . . . .	811
40.4.3	SHELL_USE_FILE_STREAM . . . . .	811
40.4.4	SHELL_AUTO_COMPLETE . . . . .	811
40.4.5	SHELL_BUFFER_SIZE . . . . .	811
40.4.6	SHELL_MAX_ARGS . . . . .	811
40.4.7	SHELL_HIST_MAX . . . . .	811
40.4.8	SHELL_MAX_CMD . . . . .	811
<b>40.5</b>	<b>Typedef Documentation</b> . . . . .	<b>811</b>
40.5.1	send_data_cb_t . . . . .	811
40.5.2	recv_data_cb_t . . . . .	811
40.5.3	printf_data_t . . . . .	811
40.5.4	cmd_function_t . . . . .	811
<b>40.6</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>811</b>
40.6.1	fun_key_status_t . . . . .	811
<b>40.7</b>	<b>Function Documentation</b> . . . . .	<b>812</b>
40.7.1	SHELL_Init . . . . .	812
40.7.2	SHELL_RegisterCommand . . . . .	812
40.7.3	SHELL_Main . . . . .	812

## Chapter DMA Manager

<b>41.1</b>	<b>Overview</b> . . . . .	<b>815</b>
<b>41.2</b>	<b>Function groups</b> . . . . .	<b>815</b>
41.2.1	DMAMGR Initialization and De-initialization . . . . .	815
41.2.2	DMAMGR Operation . . . . .	815
<b>41.3</b>	<b>Typical use case</b> . . . . .	<b>815</b>
41.3.1	DMAMGR static channel allocate . . . . .	815
41.3.2	DMAMGR dynamic channel allocate . . . . .	815
<b>41.4</b>	<b>Macro Definition Documentation</b> . . . . .	<b>816</b>
41.4.1	DMAMGR_DYNAMIC_ALLOCATE . . . . .	816
<b>41.5</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>816</b>
41.5.1	_dma_manager_status . . . . .	816
<b>41.6</b>	<b>Function Documentation</b> . . . . .	<b>816</b>
41.6.1	DMAMGR_Init . . . . .	816

# Contents

Section Number	Title	Page Number
41.6.2	DMAMGR_Deinit . . . . .	817
41.6.3	DMAMGR_RequestChannel . . . . .	817
41.6.4	DMAMGR_ReleaseChannel . . . . .	817

## **Chapter Secured Digital Card/Embedded MultiMedia Card (CARD)**

<b>42.1</b>	<b>Overview</b> . . . . .	<b>819</b>
<b>42.2</b>	<b>Data Structure Documentation</b> . . . . .	<b>822</b>
42.2.1	struct sd_card_t . . . . .	822
42.2.2	struct mmc_card_t . . . . .	823
42.2.3	struct mmc_boot_config_t . . . . .	824
<b>42.3</b>	<b>Macro Definition Documentation</b> . . . . .	<b>824</b>
42.3.1	FSL_SDMMC_DRIVER_VERSION . . . . .	824
<b>42.4</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>824</b>
42.4.1	_sdmmc_status . . . . .	824
42.4.2	_sd_card_flag . . . . .	825
42.4.3	_mmc_card_flag . . . . .	825
<b>42.5</b>	<b>Function Documentation</b> . . . . .	<b>826</b>
42.5.1	SD_Init . . . . .	826
42.5.2	SD_Deinit . . . . .	827
42.5.3	SD_CheckReadOnly . . . . .	828
42.5.4	SD_ReadBlocks . . . . .	828
42.5.5	SD_WriteBlocks . . . . .	829
42.5.6	SD_EraseBlocks . . . . .	830
42.5.7	MMC_Init . . . . .	830
42.5.8	MMC_Deinit . . . . .	831
42.5.9	MMC_CheckReadOnly . . . . .	831
42.5.10	MMC_ReadBlocks . . . . .	832
42.5.11	MMC_WriteBlocks . . . . .	832
42.5.12	MMC_EraseGroups . . . . .	833
42.5.13	MMC_SelectPartition . . . . .	834
42.5.14	MMC_SetBootConfig . . . . .	834

## **Chapter SPI based Secured Digital Card (SDSPI)**

<b>43.1</b>	<b>Overview</b> . . . . .	<b>837</b>
<b>43.2</b>	<b>Data Structure Documentation</b> . . . . .	<b>839</b>
43.2.1	struct sdspi_command_t . . . . .	839
43.2.2	struct sdspi_host_t . . . . .	839
43.2.3	struct sdspi_card_t . . . . .	839

# Contents

Section Number	Title	Page Number
<b>43.3</b>	<b>Enumeration Type Documentation</b>	<b>840</b>
43.3.1	_sdspi_status . . . . .	840
43.3.2	_sdspi_card_flag . . . . .	841
43.3.3	sdspi_response_type_t . . . . .	841
<b>43.4</b>	<b>Function Documentation</b>	<b>841</b>
43.4.1	SDSPI_Init . . . . .	841
43.4.2	SDSPI_Deinit . . . . .	842
43.4.3	SDSPI_CheckReadOnly . . . . .	842
43.4.4	SDSPI_ReadBlocks . . . . .	843
43.4.5	SDSPI_WriteBlocks . . . . .	843



# Chapter 1

## Introduction

The Kinetis Software Development Kit (KSDK) 2.0 is a collection of software enablement, for NXP Kinetis Microcontrollers, that includes peripheral drivers, high-level stacks including USB and lwIP, integration with WolfSSL and mbed TLS cryptography libraries, other middleware packages (multicore support and FatFS), and integrated RTOS support for FreeRTOS, µC/OS-II, and µC/OS-III. In addition to the base enablement, the KSDK is augmented with demo applications, driver example projects, and API documentation to help users quickly leverage the support of the Kinetis SDK. The Kinetis Expert (KEx) Web UI is available to provide access to all Kinetis SDK packages. See the *Kinetis SDK v.2.0.0 Release Notes* (document KSDK200RN) and the supported Devices section at [www.nxp.com/ksdk](http://www.nxp.com/ksdk) for details.

The Kinetis 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.
- Open-source peripheral drivers that provide stateless, high-performance, ease-of-use APIs. Communication drivers provide higher-level transactional APIs for a higher-performance option.
- Open-source RTOS wrapper driver built on top of KSDK peripheral drivers and leverage native RTOS services to better comply to the RTOS cases.
- Real time operation systems (RTOS) including FreeRTOS OS, µC/OS-II, and µC/OS-III.
- Stacks and middleware in source or object formats including:
  - A USB device, host, and OTG stack with comprehensive USB class support.
  - CMSIS-DSP, a suite of common signal processing functions.
  - FatFs, a FAT file system for small embedded systems.
  - Encryption software utilizing the mmCAU hardware acceleration.
  - SDMMC, a software component supporting SD Cards and eMMC.
  - mbedTLS, cryptographic SSL/TLS libraries.
  - lwIP, a light-weight TCP/IP stack.
  - WolfSSL, a cryptography and SSL/TLS library.
  - EMV L1 that complies to EMV-v4.3\_Book\_1 specification.
  - DMA Manager, a software component used for managing on-chip DMA channel resources.
  - The Kinetis SDK comes complete with software examples demonstrating the usage of the peripheral drivers, RTOS wrapper drivers, middleware and RTOSes.

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

- Atollic TrueSTUDIO
- GNU toolchain for ARM® Cortex® -M with Cmake build system
- IAR Embedded Workbench
- Keil MDK
- Kinetis Design Studio

The peripheral drivers and RTOS driver wrappers can be used across multiple devices within the Kinetis product family without modification. The configuration items for each driver are encapsulated into C

language data structures. Kinetis device-specific configuration information is provided as part of the KSDK 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 Kinetis SDK folder structure is organized to reduce the total number of includes required to compile a project.

<b>Deliverable</b>	<b>Location</b>
Examples	<install_dir>/examples/
Demo Applications	<install_dir>/examples/<board_name>/demo_apps/
Driver Examples	<install_dir>/examples/<board_name>/driver-examples/
Documentation	<install_dir>/docs/
USB Documentation	<install_dir>/docs/usb/
lwIP Documentation	<install_dir>/docs/tcpip/lwip/
Middleware	<install_dir>/middleware/
DMA Manager	<install_dir>/dma_manager_<version>/
FatFs	<install_dir>/middleware/fatfs_<version>
lwIP TCP/IP	<install_dir>/middleware/lwip_<version>/
mmCAU	<install_dir>/mmcau_<version>/
SDMMC Support	<install_dir>/sdmmc_<version>/
USB Stack	<install_dir>/middleware/usb_<version>
Drivers	<install_dir>/<device_name>/drivers/
CMSIS Standard ARM Cortex-M Headers, math and DSP Libraries	<install_dir>/<device_name>/CMSIS/
Device Startup and Linker	<install_dir>/<device_name>/<toolchain>/
KSDK Utilities	<install_dir>/<device_name>/utilities/
RTOS Kernels	<install_dir>/rtos/

Table 2: KSDK Folder Structure

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 Kinetis SDK documents, see the [kex.nxp.com/apidoc](http://kex.nxp.com/apidoc).

## Chapter 2

### Driver errors status

- `kStatus_DSPI_Error` = 601
- `kStatus_EDMA_QueueFull` = 5100
- `kStatus_EDMA_Busy` = 5101
- `kStatus_FLEXIO_I2S_Idle` = 2300
- `kStatus_FLEXIO_I2S_TxBusy` = 2301
- `kStatus_FLEXIO_I2S_RxBusy` = 2302
- `kStatus_FLEXIO_I2S_Error` = 2303
- `kStatus_FLEXIO_I2S_QueueFull` = 2304
- `kStatus_QSPI_Idle` = 4500
- `kStatus_QSPI_Busy` = 4501
- `kStatus_QSPI_Error` = 4502
- `kStatus_SMARTCARD_Success` = 4300
- `kStatus_SMARTCARD_TxBusy` = 4301
- `kStatus_SMARTCARD_RxBusy` = 4302
- `kStatus_SMARTCARD_NoTransferInProgress` = 4303
- `kStatus_SMARTCARD_Timeout` = 4304
- `kStatus_SMARTCARD_Initialized` = 4305
- `kStatus_SMARTCARD_PhysicalInitialized` = 4306
- `kStatus_SMARTCARD_CardNotActivated` = 4307
- `kStatus_SMARTCARD_InvalidInput` = 4308
- `kStatus_SMARTCARD_OtherError` = 4309
- `kStatus_SMC_StopAbort` = 3900
- `kStatus_NOTIFIER_ErrorNotificationBefore` = 9800
- `kStatus_NOTIFIER_ErrorNotificationAfter` = 9801
- `kStatus_DMAMGR_ChannelOccupied` = 5200
- `kStatus_DMAMGR_ChannelNotUsed` = 5201
- `kStatus_DMAMGR_NoFreeChannel` = 5202
- `kStatus_DMAMGR_ChannelNotMatchSource` = 5203



## Chapter 3

### Architectural Overview

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

#### Overview

The Kinetis 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 Kinetis SDK
5. Demo Applications based on the Kinetis SDK

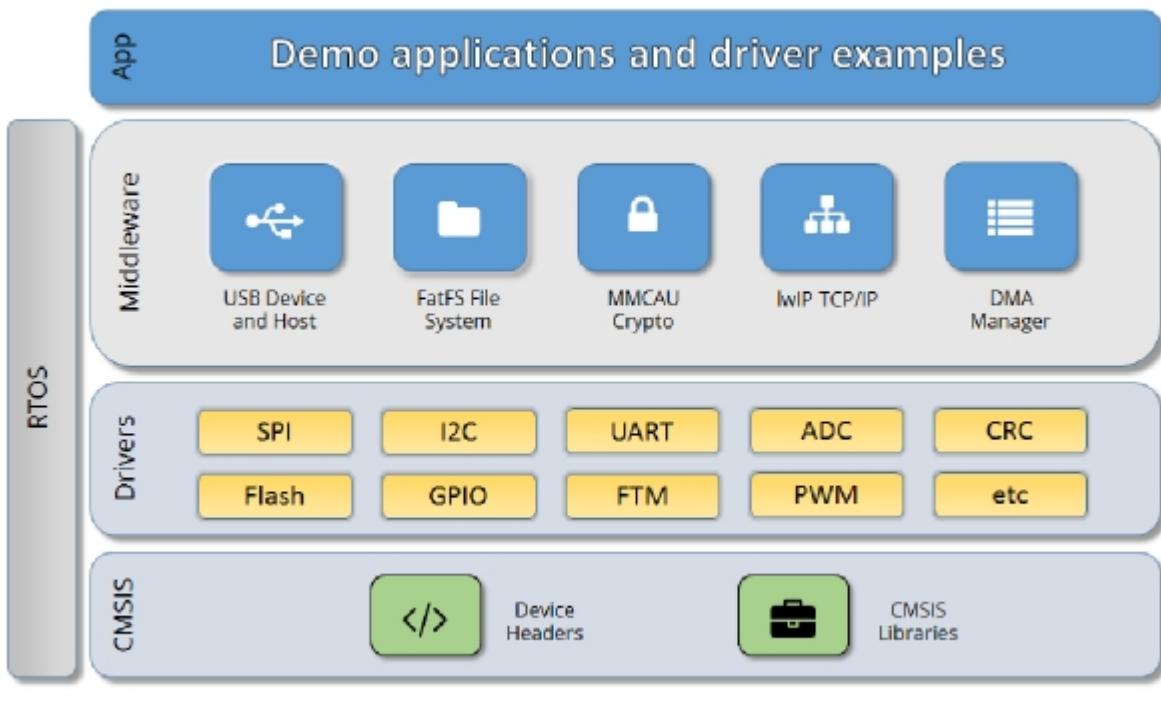


Figure 1: KSDK Block Diagram

#### Kinetis MCU header files

Each supported Kinetis MCU device in the KSDK 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 KSDK 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 KSDK 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.

## KSDK Peripheral Drivers

The KSDK peripheral drivers mainly consist of low-level functional APIs for the Kinetis MCU product family on-chip peripherals and also of high-level transactional APIs for some bus drivers/DMA driver/e-DMA driver to quickly enable the peripherals and perform transfers.

All KSDK 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 KSDK 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 Kinetis devices. It's 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 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 (BX). The KSDK drivers with transactional APIs provide the reimplementation of the second layer function inside of the peripheral driver. If the KSDK drivers with transactional APIs are linked into the image, the SPI0\_DriverIRQHandler is replaced with the function implemented in the KSDK 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 KSDK 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 Kinetis MCU device to another. An overall Peripheral Feature Header File is provided for the KSDK-supported MCU device to define the features or configuration differences for each Kinetis sub-family device.

## Application

See the *Getting Started with Kinetis SDK (KSDK) v2.0* document (KSDK20GSUG).



## **Chapter 4**

## **Trademarks**

Information in this document is provided solely to enable system and software implementers to use Freescale 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.

How to Reach Us:

Home Page: [nxp.com](http://nxp.com)

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

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale 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 Freescale 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. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: [freescale.com/SalesTermsandConditions](http://freescale.com/SalesTermsandConditions)

Freescale, the Freescale logo, Kinetis, and Processor Expert are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Tower is a trademark of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. ARM, ARM powered logo, Keil, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2016 Freescale Semiconductors, Inc.



# Chapter 5

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

### 5.1 Overview

The KSDK provides a Peripheral driver for the 16-bit SAR Analog-to-Digital Converter (ADC16) module of Kinetis devices.

### 5.2 Typical use case

#### 5.2.1 Polling Configuration

```
adc16_config_t adc16ConfigStruct;
adc16_channel_config_t adc16ChannelConfigStruct;

ADC16_Init(DEMO_ADC16_INSTANCE);
ADC16_SetDefaultConfig(&adc16ConfigStruct);
ADC16_Configure(DEMO_ADC16_INSTANCE, &adc16ConfigStruct);
ADC16_EnableHardwareTrigger(DEMO_ADC16_INSTANCE, false);
#if defined(FSL_FEATURE_ADC16_HAS_CALIBRATION) && FSL_FEATURE_ADC16_HAS_CALIBRATION
    if (kStatus_Success == ADC16_DoAutoCalibration(DEMO_ADC16_INSTANCE))
    {
        PRINTF("ADC16_DoAutoCalibration() Done.\r\n");
    }
    else
    {
        PRINTF("ADC16_DoAutoCalibration() Failed.\r\n");
    }
#endif // FSL_FEATURE_ADC16_HAS_CALIBRATION

adc16ChannelConfigStruct.channelNumber = DEMO_ADC16_USER_CHANNEL;
adc16ChannelConfigStruct.enableInterruptOnConversionCompleted =
    false;
#if defined(FSL_FEATURE_ADC16_HAS_DIFF_MODE) && FSL_FEATURE_ADC16_HAS_DIFF_MODE
    adc16ChannelConfigStruct.enableDifferentialConversion = false;
#endif // FSL_FEATURE_ADC16_HAS_DIFF_MODE

while(1)
{
    GETCHAR(); // Input any key in terminal console.
    ADC16_ChannelConfigure(DEMO_ADC16_INSTANCE, DEMO_ADC16_CHANNEL_GROUP, &adc16ChannelConfigStruct);
    while (kADC16_ChannelConversionDoneFlag !=
    ADC16_ChannelGetStatusFlags(DEMO_ADC16_INSTANCE, DEMO_ADC16_CHANNEL_GROUP))
    {
    }
    PRINTF("ADC Value: %d\r\n", ADC16_ChannelGetConversionValue(DEMO_ADC16_INSTANCE,
    DEMO_ADC16_CHANNEL_GROUP));
}
```

#### 5.2.2 Interrupt Configuration

```
volatile bool g_Adcl6ConversionDoneFlag = false;
volatile uint32_t g_Adcl6ConversionValue;
volatile uint32_t g_Adcl6InterruptCount = 0U;
```

## Typical use case

```
// ...

adc16_config_t adc16ConfigStruct;
adc16_channel_config_t adc16ChannelConfigStruct;

ADC16_Init(DEMO_ADC16_INSTANCE);
ADC16_GetDefaultConfig(&adc16ConfigStruct);
ADC16_Configure(DEMO_ADC16_INSTANCE, &adc16ConfigStruct);
ADC16_EnableHardwareTrigger(DEMO_ADC16_INSTANCE, false);
#if defined(FSL_FEATURE_ADC16_HAS_CALIBRATION) && FSL_FEATURE_ADC16_HAS_CALIBRATION
if (ADC16_DoAutoCalibration(DEMO_ADC16_INSTANCE))
{
    PRINTF("ADC16_DoAutoCalibration() Done.\r\n");
}
else
{
    PRINTF("ADC16_DoAutoCalibration() Failed.\r\n");
}
#endif // FSL_FEATURE_ADC16_HAS_CALIBRATION

adc16ChannelConfigStruct.channelNumber = DEMO_ADC16_USER_CHANNEL;
adc16ChannelConfigStruct.enableInterruptOnConversionCompleted =
    true; // Enable the interrupt.
#if defined(FSL_FEATURE_ADC16_HAS_DIFF_MODE) && FSL_FEATURE_ADC16_HAS_DIFF_MODE
adc16ChannelConfigStruct.enableDifferentialConversion = false;
#endif // FSL_FEATURE_ADC16_HAS_DIFF_MODE

while(1)
{
    GETCHAR(); // Input any key in terminal console.
    g_Adc16ConversionDoneFlag = false;
    ADC16_ChannelConfigure(DEMO_ADC16_INSTANCE, DEMO_ADC16_CHANNEL_GROUP, &adc16ChannelConfigStruct);
    while (!g_Adc16ConversionDoneFlag)
    {
    }
    PRINTF("ADC Value: %d\r\n", g_Adc16ConversionValue);
    PRINTF("ADC Interrupt Count: %d\r\n", g_Adc16InterruptCount);
}

// ...

void DEMO_ADC16_IRQHandler(void)
{
    g_Adc16ConversionDoneFlag = true;
    // Read conversion result to clear the conversion completed flag.
    g_Adc16ConversionValue = ADC16_ChannelConversionValue(DEMO_ADC16_INSTANCE, DEMO_ADC16_CHANNEL_GROUP);
    g_Adc16InterruptCount++;
}
```

## Data Structures

- struct [adc16\\_config\\_t](#)  
*ADC16 converter configuration.* [More...](#)
- struct [adc16\\_hardware\\_compare\\_config\\_t](#)  
*ADC16 Hardware compare configuration.* [More...](#)
- struct [adc16\\_channel\\_config\\_t](#)  
*ADC16 channel conversion configuration.* [More...](#)

## Enumerations

- enum [\\_adc16\\_channel\\_status\\_flags](#) { kADC16\_ChannelConversionDoneFlag = ADC\_SC1\_COCA\_O\_MASK }

- *Channel status flags.*  
• enum `_adc16_status_flags` { `kADC16_ActiveFlag` = ADC\_SC2\_ADACT\_MASK }
- *Converter status flags.*
- 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` }
   
*Converter's resolution.*
- enum `adc16_clock_source_t` {
 `kADC16_ClockSourceAlt0` = 0U,  
`kADC16_ClockSourceAlt1` = 1U,  
`kADC16_ClockSourceAlt2` = 2U,  
`kADC16_ClockSourceAlt3` = 3U,  
`kADC16_ClockSourceAsynchronousClock` = `kADC16_ClockSourceAlt3` }
   
*Clock source.*
- enum `adc16_long_sample_mode_t` {
 `kADC16_LongSampleCycle24` = 0U,  
`kADC16_LongSampleCycle16` = 1U,  
`kADC16_LongSampleCycle10` = 2U,  
`kADC16_LongSampleCycle6` = 3U,  
`kADC16_LongSampleDisabled` = 4U }
   
*Long sample mode.*
- enum `adc16_reference_voltage_source_t` {
 `kADC16_ReferenceVoltageSourceVref` = 0U,  
`kADC16_ReferenceVoltageSourceValt` = 1U }
   
*Reference voltage source.*
- enum `adc16_hardware_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, 0, 0)`)  
*ADC16 driver version 2.0.0.*

## Data Structure Documentation

### 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 converter's configuration.*

### Advanced Feature

- static void [ADC16\\_EnableHardwareTrigger](#) (ADC\_Type \*base, bool enable)  
*Enables the hardware trigger mode.*
- void [ADC16\\_SetHardwareCompareConfig](#) (ADC\_Type \*base, const adc16\_hardware\_compare\_config\_t \*config)  
*Configures the hardware compare mode.*
- uint32\_t [ADC16\\_GetStatusFlags](#) (ADC\_Type \*base)  
*Gets the status flags of the converter.*
- void [ADC16\\_ClearStatusFlags](#) (ADC\_Type \*base, uint32\_t mask)  
*Clears the status flags of the converter.*

### Conversion Channel

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

- bool `enableHighSpeed`  
*Enable the high-speed mode.*
- bool `enableLowPower`  
*Enable low power.*
- bool `enableContinuousConversion`  
*Enable continuous conversion mode.*

### 5.3.1.0.0.1 Field Documentation

5.3.1.0.0.1.1 `adc16_reference_voltage_source_t adc16_config_t::referenceVoltageSource`

5.3.1.0.0.1.2 `adc16_clock_source_t adc16_config_t::clockSource`

5.3.1.0.0.1.3 `bool adc16_config_t::enableAsynchronousClock`

5.3.1.0.0.1.4 `adc16_clock_divider_t adc16_config_t::clockDivider`

5.3.1.0.0.1.5 `adc16_resolution_t adc16_config_t::resolution`

5.3.1.0.0.1.6 `adc16_long_sample_mode_t adc16_config_t::longSampleMode`

5.3.1.0.0.1.7 `bool adc16_config_t::enableHighSpeed`

5.3.1.0.0.1.8 `bool adc16_config_t::enableLowPower`

5.3.1.0.0.1.9 `bool adc16_config_t::enableContinuousConversion`

### 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.*

### 5.3.2.0.0.2 Field Documentation

5.3.2.0.0.2.1 `adc16_hardware_compare_mode_t adc16_hardware_compare_config_t::hardwareCompareMode`

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

## Enumeration Type Documentation

5.3.2.0.0.2.2 `int16_t adc16_hardware_compare_config_t::value1`

5.3.2.0.0.2.3 `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.*

#### 5.3.3.0.0.3 Field Documentation

5.3.3.0.0.3.1 `uint32_t adc16_channel_config_t::channelNumber`

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

5.3.3.0.0.3.2 `bool adc16_channel_config_t::enableInterruptOnConversionCompleted`

## 5.4 Macro Definition Documentation

5.4.1 `#define FSL_ADC16_DRIVER_VERSION(MAKE_VERSION(2, 0, 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.

### 5.5.3 `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.

#### 5.5.4 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.

#### 5.5.5 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.

#### 5.5.6 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.

#### 5.5.7 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.

## Function Documentation

### 5.5.8 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 an available settings. The default values are:

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

Parameters

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

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

Parameters

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

#### 5.6.5 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 hardware. Only the result in compare range is available. To compare the range, see "adc16\_hardware\_compare\_mode\_t", or the reference manual document for more detailed information.

Parameters

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

#### 5.6.6 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".

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

## Function Documentation

### Parameters

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

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

This operation triggers the conversion if 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 can have more than one group of status and control register, one for each conversion. The channel group parameter indicates which group of registers are used 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. Channel group 0 is used for both software and hardware trigger modes of operation. Channel groups 1 and greater indicate potentially multiple channel group registers for use only in hardware trigger mode. See the chip configuration information in the MCU reference manual about the number of SC1n registers (channel groups) specific to this device. None of the channel groups 1 or greater are used for software trigger operation and therefore writes to these channel groups do not initiate a new conversion. Updating 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 "adc16_channel_config_t" structure for conversion channel.

### **5.6.9 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.

### 5.6.10 `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`".

## Function Documentation

# Chapter 6

## Clock Driver

### 6.1 Overview

The KSDK provides APIs for Kinetis devices clock operation.

### 6.2 Get frequency

A centralized function `CLOCK_GetFreq` gets different clock type frequencies by passing a clock name. For example, pass a `kCLOCK_CoreSysClk` to get the core clock and pass a `kCLOCK_BusClk` to get the bus clock. Additionally, there are separate functions to get frequency, for example, use `CLOCK_GetCoreSysClkFreq` to get the core clock frequency and `CLOCK_GetBusClkFreq` to get the bus clock frequency. Using these functions reduces the image size.

### 6.3 External clock frequency

The external clocks EXTAL0/EXTAL1/EXTAL32 are decided by the board level design. The Clock driver uses variables `g_xtal0Freq/g_xtal1Freq/g_xtal32Freq` to save clock frequencies. Likewise, the APIs `CLOCK_SetXtal0Freq`, `CLOCK_SetXtal1Freq` and `CLOCK_SetXtal32Freq` are used to set these variables.

The upper layer must set these values correctly, for example, after `OSC0(SYSOSC)` is initialized using `CLOCK_InitOsc0` or `CLOCK_InitSysOsc`, the upper layer should call the `CLOCK_SetXtal0Freq`. Otherwise, the clock frequency get functions may not get valid values. This is useful for multicore platforms where only one core calls `CLOCK_InitOsc0` to initialize `OSC0` and other cores call `CLOCK_SetXtal0Freq`.

## 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...](#)

## External clock frequency

### Macros

- #define **MCG\_INTERNAL\_IRC\_48M** 48000000U  
*IRC48M clock frequency in Hz.*
- #define **DMAMUX\_CLOCKS**  
*Clock ip name array for DMAMUX.*
- #define **RTC\_CLOCKS**  
*Clock ip name array for RTC.*
- #define **DRYICE\_CLOCKS**  
*Clock ip name array for DRYICE.*
- #define **PORT\_CLOCKS**  
*Clock ip name array for PORT.*
- #define **EWM\_CLOCKS**  
*Clock ip name array for EWM.*
- #define **PIT\_CLOCKS**  
*Clock ip name array for PIT.*
- #define **DSPI\_CLOCKS**  
*Clock ip name array for DSPI.*
- #define **EMVSIM\_CLOCKS**  
*Clock ip name array for EMVSIM.*
- #define **QSPI\_CLOCKS**  
*Clock ip name array for QSPI.*
- #define **EDMA\_CLOCKS**  
*Clock ip name array for EDMA.*
- #define **LPUART\_CLOCKS**  
*Clock ip name array for LPUART.*
- #define **DAC\_CLOCKS**  
*Clock ip name array for DAC.*
- #define **LPTMR\_CLOCKS**  
*Clock ip name array for LPTMR.*
- #define **ADC16\_CLOCKS**  
*Clock ip name array for ADC16.*
- #define **TRNG\_CLOCKS**  
*Clock ip name array for TRNG.*
- #define **MPU\_CLOCKS**  
*Clock ip name array for MPU.*
- #define **FLEXIO\_CLOCKS**  
*Clock ip name array for FLEXIO.*
- #define **VREF\_CLOCKS**  
*Clock ip name array for VREF.*
- #define **TPM\_CLOCKS**  
*Clock ip name array for TPM.*
- #define **TSI\_CLOCKS**  
*Clock ip name array for TSI.*
- #define **LTC\_CLOCKS**  
*Clock ip name array for LTC.*
- #define **CRC\_CLOCKS**  
*Clock ip name array for CRC.*
- #define **I2C\_CLOCKS**  
*Clock ip name array for I2C.*
- #define **CMP\_CLOCKS**  
*Clock ip name array for CMP.*

- #define **INTMUX\_CLOCKS**  
*Clock ip name array for INTMUX.*
- #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\_FastPeriphClk**,  
**kCLOCK\_PllFllSelClk**,  
**kCLOCK\_QspiBusClk**,  
**kCLOCK\_Er32kClk**,  
**kCLOCK\_Osc0ErClk**,  
**kCLOCK\_Osc1ErClk**,  
**kCLOCK\_Osc0ErClkUndiv**,  
**kCLOCK\_McgFixedFreqClk**,  
**kCLOCK\_McgInternalRefClk**,  
**kCLOCK\_McgFllClk**,  
**kCLOCK\_McgPll0Clk**,  
**kCLOCK\_McgPll1Clk**,  
**kCLOCK\_McgExtPllClk**,  
**kCLOCK\_McgPeriphClk**,  
**kCLOCK\_McgIrc48MClk**,  
**kCLOCK\_LpoClk** }
- Clock name used to get clock frequency.*
- enum **clock\_usb\_src\_t** {
 **kCLOCK\_UsbSrcPll0** = SIM\_SOPT2\_USBSRC(1U) | SIM\_SOPT2\_PLLFLLSEL(1U),  
**kCLOCK\_UsbSrcIrc48M** = SIM\_SOPT2\_USBSRC(1U) | SIM\_SOPT2\_PLLFLLSEL(3U),  
**kCLOCK\_UsbSrcExt** = SIM\_SOPT2\_USBSRC(0U) }
- USB clock source definition.*
- enum **clock\_ip\_name\_t**  
*Clock gate name used for CLOCK\_EnableClock/CLOCK\_DisableClock.*
- enum **osc\_mode\_t** {
 **kOSC\_ModeExt** = 0U,  
**kOSC\_ModeOscLowPower** = MCG\_C2\_EREFS0\_MASK,  
**kOSC\_ModeOscHighGain** }
- OSC work mode.*
- enum **\_osc\_cap\_load** {
 **kOSC\_Cap2P** = OSC\_CR\_SC2P\_MASK,  
**kOSC\_Cap4P** = OSC\_CR\_SC4P\_MASK,  
**kOSC\_Cap8P** = OSC\_CR\_SC8P\_MASK,

## External clock frequency

- ```
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\_PliRefOsc0,  
    kMCG\_PliRefOsc1 }
- MCG PLL reference clock select.*
- enum `mcg_clkout_src_t` {  
    kMCG\_ClkOutSrcOut,  
    kMCG\_ClkOutSrcInternal,  
    kMCG\_ClkOutSrcExternal }
- MCGOUT clock source.*
- enum `mcg_atm_select_t` {  
    kMCG\_AtmSel32k,  
    kMCG\_AtmSel4m }
- MCG Automatic Trim Machine Select.*
- enum `mcg_oscsel_t` {  
    kMCG\_OscselOsc,  
    kMCG\_OscselRtc,  
    kMCG\_OscselIrc }
- MCG OSC Clock Select.*
- enum `mcg_pll_clk_select_t` { kMCG\_PliClkSelPli0 }
- 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_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_SetEmvsimClock` (`uint32_t` src)

## External clock frequency

- static void **CLOCK\_SetLpuartClock** (uint32\_t src)  
*Set LPUART clock source.*
- static void **CLOCK\_SetTpmClock** (uint32\_t src)  
*Set TPM clock source.*
- static void **CLOCK\_SetFlexio0Clock** (uint32\_t src)  
*Set FLEXIO clock source.*
- static void **CLOCK\_SetPllFllSelClock** (uint32\_t src, uint32\_t divValue, uint32\_t fracValue)  
*Set PLLFLLSEL clock source.*
- static void **CLOCK\_SetClkOutClock** (uint32\_t src)  
*Set CLKOUT source.*
- static void **CLOCK\_SetRtcClkOutClock** (uint32\_t src)  
*Set RTC\_CLKOUT source.*
- bool **CLOCK\_EnableUsbfs0Clock** (clock\_usb\_src\_t src, uint32\_t freq)  
*Enable USB FS clock.*
- static void **CLOCK\_DisableUsbfs0Clock** (void)  
*Disable USB FS clock.*
- static void **CLOCK\_SetOutDiv** (uint32\_t outdiv1, uint32\_t outdiv2, uint32\_t outdiv4, uint32\_t outdiv5)  
*System clock divider.*
- uint32\_t **CLOCK\_GetFreq** (clock\_name\_t clockName)  
*Gets the clock frequency for a specific clock name.*
- uint32\_t **CLOCK\_GetCoreSysClkFreq** (void)  
*Get the core clock or system clock frequency.*
- uint32\_t **CLOCK\_GetPlatClkFreq** (void)  
*Get the platform clock frequency.*
- uint32\_t **CLOCK\_GetBusClkFreq** (void)  
*Get the bus clock frequency.*
- uint32\_t **CLOCK\_GetFlashClkFreq** (void)  
*Get the flash clock frequency.*
- uint32\_t **CLOCK\_GetPllFllSelClkFreq** (void)  
*Get the output clock frequency selected by SIM[PLLFLLSEL].*
- uint32\_t **CLOCK\_GetQspiBusClkFreq** (void)  
*Get the QSPI bus interface clock frequency.*
- uint32\_t **CLOCK\_GetEr32kClkFreq** (void)  
*Get the external reference 32K clock frequency (ERCLK32K).*
- uint32\_t **CLOCK\_GetOsc0ErClkUndivFreq** (void)  
*Get the OSC0 external reference undivided clock frequency (OSC0ERCLK\_UNDIV).*
- uint32\_t **CLOCK\_GetOsc0ErClkFreq** (void)  
*Get the OSC0 external reference clock frequency (OSC0ERCLK).*
- 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

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

## Driver version

- #define **FSL\_CLOCK\_DRIVER\_VERSION** (**MAKE\_VERSION**(2, 2, 0))  
*CLOCK driver version 2.2.0.*

## 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 fcr-div)**  
*Configures the Internal Reference clock (MCGIRCLK).*
- status\_t **CLOCK\_SetExternalRefClkConfig (mcg\_oscsel\_t oscsel)**  
*Selects the MCG external reference clock.*
- void **CLOCK\_EnablePll0 (mcg\_pll\_config\_t const \*config)**  
*Enables the PLL0 in FLL mode.*
- static void **CLOCK\_DisablePll0 (void)**  
*Disables the PLL0 in FLL mode.*
- uint32\_t **CLOCK\_CalcPllDiv (uint32\_t refFreq, uint32\_t desireFreq, uint8\_t \*prdiv, uint8\_t \*vdiv)**  
*Calculates the PLL divider setting for a desired output frequency.*

## MCG clock lock monitor functions.

- void **CLOCK\_SetOsc0MonitorMode (mcg\_monitor\_mode\_t mode)**  
*Sets the OSC0 clock monitor mode.*
- void **CLOCK\_SetRtcOscMonitorMode (mcg\_monitor\_mode\_t mode)**  
*Sets the RTC OSC clock monitor mode.*
- void **CLOCK\_SetPll0MonitorMode (mcg\_monitor\_mode\_t mode)**  
*Sets the PLL0 clock monitor mode.*
- uint32\_t **CLOCK\_GetStatusFlags (void)**  
*Gets the MCG status flags.*
- void **CLOCK\_ClearStatusFlags (uint32\_t mask)**  
*Clears the MCG status flags.*

## OSC configuration

- static void **OSC\_SetExtRefClkConfig (OSC\_Type \*base, oscer\_config\_t const \*config)**  
*Configures the OSC external reference clock (OSCERCLK).*

## External clock frequency

- 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.*

## MCG auto-trim machine.

- status\_t **CLOCK\_TrimInternalRefClk** (uint32\_t extFreq, uint32\_t desireFreq, uint32\_t \*actualFreq, mcg\_atm\_select\_t atms)  
*Auto trims the internal reference clock.*

## MCG mode functions.

- mcg\_mode\_t **CLOCK\_GetMode** (void)  
*Gets the current MCG mode.*
- status\_t **CLOCK\_SetFeiMode** (mcg\_dmx32\_t dmx32, mcg\_drs\_t drs, void(\*fllStableDelay)(void))  
*Sets the MCG to FEI mode.*
- status\_t **CLOCK\_SetFeeMode** (uint8\_t frdiv, mcg\_dmx32\_t dmx32, mcg\_drs\_t drs, void(\*fllStableDelay)(void))  
*Sets the MCG to FEE mode.*
- status\_t **CLOCK\_SetFbiMode** (mcg\_dmx32\_t dmx32, mcg\_drs\_t drs, void(\*fllStableDelay)(void))  
*Sets the MCG to FBI mode.*
- status\_t **CLOCK\_SetFbeMode** (uint8\_t frdiv, mcg\_dmx32\_t dmx32, mcg\_drs\_t drs, void(\*fllStableDelay)(void))  
*Sets the MCG to FBE mode.*
- status\_t **CLOCK\_SetBlpiMode** (void)  
*Sets the MCG to BLPI mode.*
- status\_t **CLOCK\_SetBlpeMode** (void)  
*Sets the MCG to BLPE mode.*
- status\_t **CLOCK\_SetPbeMode** (mcg\_pll\_clk\_select\_t pllcs, mcg\_pll\_config\_t const \*config)  
*Sets the MCG to PBE mode.*
- status\_t **CLOCK\_SetPeeMode** (void)  
*Sets the MCG to PEE mode.*
- status\_t **CLOCK\_ExternalModeToFbeModeQuick** (void)  
*Switches the MCG to FBE mode from the external mode.*
- status\_t **CLOCK\_InternalModeToFbiModeQuick** (void)  
*Switches the MCG to FBI mode from internal modes.*
- status\_t **CLOCK\_BootToFeiMode** (mcg\_dmx32\_t dmx32, mcg\_drs\_t drs, void(\*fllStableDelay)(void))  
*Sets the MCG to FEI mode during system boot up.*

- status\_t **CLOCK\_BootToFeeMode** (**mcg\_oscsel\_t** oscsel, uint8\_t frdiv, **mcg\_dmx32\_t** dmx32, **mcg\_drs\_t** drs, void(\*flStableDelay)(void))  
*Sets the MCG to FEE mode during system bootup.*
- status\_t **CLOCK\_BootToBlpiMode** (uint8\_t fcrdiv, **mcg\_irc\_mode\_t** ircs, uint8\_t ircEnableMode)  
*Sets the MCG to BLPI mode during system boot up.*
- status\_t **CLOCK\_BootToBlpeMode** (**mcg\_oscsel\_t** oscsel)  
*Sets the MCG to BLPE mode during system boot up.*
- status\_t **CLOCK\_BootToPeeMode** (**mcg\_oscsel\_t** oscsel, **mcg\_pll\_clk\_select\_t** pllcs, **mcg\_pll\_config\_t** const \*config)  
*Sets the MCG to PEE mode during system boot up.*
- status\_t **CLOCK\_SetMcgConfig** (**mcg\_config\_t** const \*config)  
*Sets the MCG to a target mode.*

## 6.4 Data Structure Documentation

### 6.4.1 struct sim\_clock\_config\_t

#### Data Fields

- uint8\_t **pllFllSel**  
*PLL/FLL/IRC48M selection.*
- uint8\_t **pllFllDiv**  
*PLL/LLSEL clock divider divisor.*
- uint8\_t **pllFllFrac**  
*PLL/LLSEL clock divider fraction.*
- uint8\_t **er32kSrc**  
*ERCLK32K source selection.*
- uint32\_t **clkdiv1**  
*SIM\_CLKDIV1.*

#### 6.4.1.0.0.4 Field Documentation

- 6.4.1.0.0.4.1 uint8\_t **sim\_clock\_config\_t::pllFllSel**
- 6.4.1.0.0.4.2 uint8\_t **sim\_clock\_config\_t::pllFllDiv**
- 6.4.1.0.0.4.3 uint8\_t **sim\_clock\_config\_t::pllFllFrac**
- 6.4.1.0.0.4.4 uint8\_t **sim\_clock\_config\_t::er32kSrc**
- 6.4.1.0.0.4.5 uint32\_t **sim\_clock\_config\_t::clkdiv1**

### 6.4.2 struct oscer\_config\_t

#### Data Fields

- uint8\_t **enableMode**  
*OSCERCLK enable mode.*
- uint8\_t **erclkDiv**

## Data Structure Documentation

*Divider for OSCERCLK.*

### 6.4.2.0.0.5 Field Documentation

#### 6.4.2.0.0.5.1 uint8\_t oscer\_config\_t::enableMode

OR'ed value of [\\_oscer\\_enable\\_mode](#).

#### 6.4.2.0.0.5.2 uint8\_t oscer\_config\_t::erclkDiv

### 6.4.3 struct osc\_config\_t

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

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

## Data Fields

- [uint32\\_t freq](#)  
*External clock frequency.*
- [uint8\\_t capLoad](#)  
*Capacitor load setting.*
- [osc\\_mode\\_t workMode](#)  
*OSC work mode setting.*
- [oscer\\_config\\_t oscerConfig](#)  
*Configuration for OSCERCLK.*

### 6.4.3.0.0.6 Field Documentation

#### 6.4.3.0.0.6.1 uint32\_t osc\_config\_t::freq

#### 6.4.3.0.0.6.2 uint8\_t osc\_config\_t::capLoad

#### 6.4.3.0.0.6.3 osc\_mode\_t osc\_config\_t::workMode

#### 6.4.3.0.0.6.4 oscer\_config\_t osc\_config\_t::oscerConfig

### 6.4.4 struct mcg\_pll\_config\_t

## Data Fields

- [uint8\\_t enableMode](#)  
*Enable mode.*
- [uint8\\_t prdiv](#)  
*Reference divider PRDIV.*
- [uint8\\_t vdiv](#)

*VCO divider VDIV.*

#### 6.4.4.0.0.7 Field Documentation

##### 6.4.4.0.0.7.1 uint8\_t mcg\_pll\_config\_t::enableMode

OR'ed value of [\\_mcg\\_pll\\_enable\\_mode](#).

##### 6.4.4.0.0.7.2 uint8\_t mcg\_pll\_config\_t::prdiv

##### 6.4.4.0.0.7.3 uint8\_t mcg\_pll\_config\_t::vdiv

#### 6.4.5 struct mcg\_config\_t

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

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

#### Data Fields

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

## Macro Definition Documentation

### 6.4.5.0.0.8 Field Documentation

6.4.5.0.0.8.1 `mcg_mode_t mcg_config_t::mcgMode`

6.4.5.0.0.8.2 `uint8_t mcg_config_t::irclkEnableMode`

6.4.5.0.0.8.3 `mcg_irc_mode_t mcg_config_t::ircs`

6.4.5.0.0.8.4 `uint8_t mcg_config_t::fcrdiv`

6.4.5.0.0.8.5 `uint8_t mcg_config_t::frdiv`

6.4.5.0.0.8.6 `mcg_drs_t mcg_config_t::drs`

6.4.5.0.0.8.7 `mcg_dmx32_t mcg_config_t::dmx32`

6.4.5.0.0.8.8 `mcg_oscsel_t mcg_config_t::oscsel`

6.4.5.0.0.8.9 `mcg_pll_config_t mcg_config_t::pll0Config`

## 6.5 Macro Definition Documentation

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

6.5.2 `#define MCG_INTERNAL_IRC_48M 48000000U`

6.5.3 `#define DMAMUX_CLOCKS`

**Value:**

```
{           \
    kCLOCK_Dmamux0 \
}
```

6.5.4 `#define RTC_CLOCKS`

**Value:**

```
{           \
    kCLOCK_Rtc0 \
}
```

6.5.5 `#define DRYICE_CLOCKS`

**Value:**

```
{  
    kCLOCK_Dryice0 \  
}
```

## 6.5.6 #define PORT\_CLOCKS

**Value:**

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

## 6.5.7 #define EWM\_CLOCKS

**Value:**

```
{  
    kCLOCK_Ewm0 \  
}
```

## 6.5.8 #define PIT\_CLOCKS

**Value:**

```
{  
    kCLOCK_Pit0 \  
}
```

## 6.5.9 #define DSPI\_CLOCKS

**Value:**

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

## 6.5.10 #define EMVSIM\_CLOCKS

**Value:**

```
{  
    kCLOCK_Emvsim0, kCLOCK_Emvsim1 \  
}
```

## Macro Definition Documentation

### 6.5.11 #define QSPI\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Qspi0 \
}
```

### 6.5.12 #define EDMA\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Dma0 \
}
```

### 6.5.13 #define LPUART\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Lpuart0, kCLOCK_Lpuart1, kCLOCK_Lpuart2 \
}
```

### 6.5.14 #define DAC\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Dac0 \
}
```

### 6.5.15 #define LPTMR\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Lptmr0, kCLOCK_Lptmr1 \
}
```

### 6.5.16 #define ADC16\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Adc0  \
}
```

### 6.5.17 #define TRNG\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Trng0 \
}
```

### 6.5.18 #define MPU\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Mpu0 \
}
```

### 6.5.19 #define FLEXIO\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Flexio0 \
}
```

### 6.5.20 #define VREF\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Vref0 \
}
```

## Macro Definition Documentation

### 6.5.21 #define TPM\_CLOCKS

**Value:**

```
{  
    kCLOCK_Tpm0, kCLOCK_Tpm1, kCLOCK_Tpm2 \  
}
```

### 6.5.22 #define TSI\_CLOCKS

**Value:**

```
{  
    kCLOCK_Tsi0 \  
}
```

### 6.5.23 #define LTC\_CLOCKS

**Value:**

```
{  
    kCLOCK_Ltc0 \  
}
```

### 6.5.24 #define CRC\_CLOCKS

**Value:**

```
{  
    kCLOCK_Crc0 \  
}
```

### 6.5.25 #define I2C\_CLOCKS

**Value:**

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

### 6.5.26 #define CMP\_CLOCKS

**Value:**

```
{
    \_kCLOCK_Cmp0 \
}
```

### 6.5.27 #define INTMUX\_CLOCKS

**Value:**

```
{
    \_kCLOCK_Intmux0 \
}
```

### 6.5.28 #define SYS\_CLK kCLOCK\_CoreSysClk

## 6.6 Enumeration Type Documentation

### 6.6.1 enum clock\_name\_t

Enumerator

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

## Enumeration Type Documentation

### 6.6.2 enum clock\_usb\_src\_t

Enumerator

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

### 6.6.3 enum clock\_ip\_name\_t

### 6.6.4 enum osc\_mode\_t

Enumerator

- kOSC\_ModeExt* Use an external clock.
- kOSC\_ModeOscLowPower* Oscillator low power.
- kOSC\_ModeOscHighGain* Oscillator high gain.

### 6.6.5 enum \_osc\_cap\_load

Enumerator

- kOSC\_Cap2P* 2 pF capacitor load
- kOSC\_Cap4P* 4 pF capacitor load
- kOSC\_Cap8P* 8 pF capacitor load
- kOSC\_Cap16P* 16 pF capacitor load

### 6.6.6 enum \_oscer\_enable\_mode

Enumerator

- kOSC\_ErClkEnable* Enable.
- kOSC\_ErClkEnableInStop* Enable in stop mode.

### 6.6.7 enum mcg\_fll\_src\_t

Enumerator

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

**6.6.8 enum mcg\_irc\_mode\_t**

Enumerator

*kMCG\_IrcSlow* Slow internal reference clock selected.*kMCG\_IrcFast* Fast internal reference clock selected.**6.6.9 enum mcg\_dmx32\_t**

Enumerator

*kMCG\_Dmx32Default* DCO has a default range of 25%.*kMCG\_Dmx32Fine* DCO is fine-tuned for maximum frequency with 32.768 kHz reference.**6.6.10 enum mcg\_drs\_t**

Enumerator

*kMCG\_DrsLow* Low frequency range.*kMCG\_DrsMid* Mid frequency range.*kMCG\_DrsMidHigh* Mid-High frequency range.*kMCG\_DrsHigh* High frequency range.**6.6.11 enum mcg\_pll\_ref\_src\_t**

Enumerator

*kMCG\_PllRefOsc0* Selects OSC0 as PLL reference clock.*kMCG\_PllRefOsc1* Selects OSC1 as PLL reference clock.**6.6.12 enum mcg\_clkout\_src\_t**

Enumerator

*kMCG\_ClkOutSrcOut* Output of the FLL is selected (reset default)*kMCG\_ClkOutSrcInternal* Internal reference clock is selected.*kMCG\_ClkOutSrcExternal* External reference clock is selected.

## Enumeration Type Documentation

### 6.6.13 enum mcg\_atm\_select\_t

Enumerator

*kMCG\_AtmSel32k* 32 kHz Internal Reference Clock selected

*kMCG\_AtmSel4m* 4 MHz Internal Reference Clock selected

### 6.6.14 enum mcg\_oscsel\_t

Enumerator

*kMCG\_OscselOsc* Selects System Oscillator (OSCCLK)

*kMCG\_OscselRtc* Selects 32 kHz RTC Oscillator.

*kMCG\_OscselIrc* Selects 48 MHz IRC Oscillator.

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

Enumerator

*kMCG\_PllClkSelPll0* PLL0 output clock is selected.

### 6.6.16 enum mcg\_monitor\_mode\_t

Enumerator

*kMCG\_MonitorNone* Clock monitor is disabled.

*kMCG\_MonitorInt* Trigger interrupt when clock lost.

*kMCG\_MonitorReset* System reset when clock lost.

### 6.6.17 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.

**6.6.18 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.

**6.6.19 enum \_mcg\_irclk\_enable\_mode**

Enumerator

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

**6.6.20 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.

**6.6.21 enum mcg\_mode\_t**

Enumerator

*kMCG\_ModeFEI* FEI - FLL Engaged Internal.  
*kMCG\_ModeFBI* FBI - FLL Bypassed Internal.  
*kMCG\_ModeBLPI* BLPI - Bypassed Low Power Internal.  
*kMCG\_ModeFEE* FEE - FLL Engaged External.  
*kMCG\_ModeFBE* FBE - FLL Bypassed External.  
*kMCG\_ModeBLPE* BLPE - Bypassed Low Power External.  
*kMCG\_ModePBE* PBE - PLL Bypassed External.  
*kMCG\_ModePEE* PEE - PLL Engaged External.  
*kMCG\_ModeError* Unknown mode.

## Function Documentation

### 6.7 Function Documentation

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

Parameters

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

### 6.7.2 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> . |
|-------------|---------------------------------------------------------------|

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

Parameters

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

### 6.7.4 static void CLOCK\_SetEmvsimClock( *uint32\_t src* ) [inline], [static]

Parameters

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

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

Parameters

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

### 6.7.6 static void CLOCK\_SetTpmClock( *uint32\_t src* ) [inline], [static]

## Function Documentation

Parameters

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

### 6.7.7 static void CLOCK\_SetFlexio0Clock ( uint32\_t *src* ) [inline], [static]

Parameters

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

### 6.7.8 static void CLOCK\_SetPllFllSelClock ( uint32\_t *src*, uint32\_t *divValue*, uint32\_t *fracValue* ) [inline], [static]

Parameters

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

Return values

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

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

Disable USB FS clock.

### 6.7.13 static void CLOCK\_SetOutDiv( uint32\_t *outdiv1*, uint32\_t *outdiv2*, uint32\_t *outdiv4*, uint32\_t *outdiv5* ) [inline], [static]

Set the SIM\_CLKDIV1[OUTDIV1], SIM\_CLKDIV1[OUTDIV2], SIM\_CLKDIV1[OUTDIV4], SIM\_CLKDIV1[OUTDIV5].

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>outdiv1</i> | Clock 1 output divider value. |
| <i>outdiv2</i> | Clock 2 output divider value. |
| <i>outdiv4</i> | Clock 4 output divider value. |
| <i>outdiv5</i> | Clock 5 output divider value. |

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

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

Parameters

## Function Documentation

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

Returns

Clock frequency value in Hertz

### **6.7.15 `uint32_t CLOCK_GetCoreSysClkFreq( void )`**

Returns

Clock frequency in Hz.

### **6.7.16 `uint32_t CLOCK_GetPlatClkFreq( void )`**

Returns

Clock frequency in Hz.

### **6.7.17 `uint32_t CLOCK_GetBusClkFreq( void )`**

Returns

Clock frequency in Hz.

### **6.7.18 `uint32_t CLOCK_GetFlashClkFreq( void )`**

Returns

Clock frequency in Hz.

### **6.7.19 `uint32_t CLOCK_GetPIIFIISelClkFreq( void )`**

Returns

Clock frequency in Hz.

**6.7.20 uint32\_t CLOCK\_GetQspiBusClkFreq( void )**

Returns

Clock frequency in Hz.

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

Returns

Clock frequency in Hz.

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

Returns

Clock frequency in Hz.

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

Returns

Clock frequency in Hz.

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

This function sets system layer clock settings in SIM module.

Parameters

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

**6.7.25 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.

## Function Documentation

Parameters

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

### **6.7.26 uint32\_t CLOCK\_GetOutClkFreq ( void )**

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

Returns

The frequency of MCGOUTCLK.

### **6.7.27 uint32\_t CLOCK\_GetFllFreq ( void )**

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

Returns

The frequency of MCGFLLCLK.

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

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

Returns

The frequency of MCGIRCLK.

### **6.7.29 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.

**6.7.30 uint32\_t CLOCK\_GetPLL0Freq( void )**

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

Returns

The frequency of MCGPLL0CLK.

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

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

Parameters

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

**6.7.32 status\_t CLOCK\_SetInternalRefClkConfig( uint8\_t enableMode, mcg\_ircc\_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

## Function Documentation

|                               |                                                                                                                                      |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <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.                                                                                        |

### 6.7.33 **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.                                                                                           |

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

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

Parameters

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

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

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

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

This function calculates the correct reference clock divider (PRDIV) and VCO divider (VDIV) to generate a desired PLL output frequency. It returns the closest frequency match with the corresponding PRDIV/-

VDIV returned from parameters. If a desired frequency is not valid, this function returns 0.

## Function Documentation

Parameters

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

Returns

Closest frequency match that the PLL was able generate.

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

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

Parameters

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

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

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

Parameters

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

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

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

Parameters

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

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

This function gets the MCG clock status flags. All status flags are returned as a logical OR of the enumeration [\\_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](#).

#### 6.7.41 void CLOCK\_ClearStatusFlags ( uint32\_t *mask* )

This function clears the MCG clock lock lost status. The parameter is a logical OR value of the flags to clear. See [\\_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> . |
|-------------|---------------------------------------------------------------------------------------------------------------------|

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

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

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

OSC_SetExtRefClkConfig(OSC, &config);
```

## Function Documentation

Parameters

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

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

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

Parameters

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

Example:

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

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

This function initializes the OSC0 according to the board configuration.

Parameters

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

### 6.7.45 void CLOCK\_DeinitOsc0 ( void )

This function deinitializes the OSC0.

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

Parameters

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

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

Parameters

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

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

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

Parameters

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

Return values

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

## Function Documentation

|                                     |                                |
|-------------------------------------|--------------------------------|
| <i>kStatus_MCG_Atm-HardwareFail</i> | Hardware fails while trimming. |
|-------------------------------------|--------------------------------|

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

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

Returns

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

### 6.7.50 **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.

### 6.7.51 **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. |

### 6.7.52 **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. |

## Function Documentation

### Note

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

### 6.7.53 **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. |

**6.7.54 status\_t CLOCK\_SetBlpiMode( void )**

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

## Return values

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

**6.7.55 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. |
|-------------------------------------|--------------------------------------|

## Function Documentation

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

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

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

Parameters

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

Return values

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

Note

1. The parameter *pllcs* selects the PLL. For platforms with only one PLL, the parameter *pllcs* is kept for interface compatibility.
2. The parameter *config* is the PLL configuration structure. On some platforms, it is possible to choose the external PLL directly, which renders the configuration structure not necessary. In this case, pass in NULL. For example: CLOCK\_SetPbeMode(kMCG\_OscselOsc, kMCG\_PlClkSelExtPll, NULL);

### 6.7.57 **status\_t CLOCK\_SetPeeMode ( void )**

This function sets the MCG to PEE mode.

Return values

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

## Note

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

### 6.7.58 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. |

### 6.7.59 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. |

## Function Documentation

**6.7.60 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.

### 6.7.61 **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. |

## Function Documentation

**6.7.62 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. |

### 6.7.63 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. |

### 6.7.64 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]. |
|---------------|-----------------------------------|

## Variable Documentation

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

Return values

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

### 6.7.65 **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.

## 6.8 Variable Documentation

### 6.8.1 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:

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

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

## 6.8.2 `uint32_t g_xtal32Freq`

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

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

## Multipurpose Clock Generator (MCG)

### 6.9 Multipurpose Clock Generator (MCG)

The KSDK provides a peripheral driver for the MCG module of Kinetis devices.

#### 6.9.1 Function description

MCG driver provides these functions:

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

##### 6.9.1.1 MCG frequency functions

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

##### 6.9.1.2 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 micro seconds 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.

### 6.9.1.3 MCG clock lock monitor functions

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

### 6.9.1.4 OSC configuration

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

### 6.9.1.5 MCG auto-trim machine

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

### 6.9.1.6 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 case.

## Multipurpose Clock Generator (MCG)

### 6.9.2 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. To enable the corresponding clock before using it as a clock source.

#### 6.9.2.1 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>                               |

#### 6.9.2.2 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>                               |

### 6.9.2.3 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)                 |

### 6.9.2.4 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(...) |

## Multipurpose Clock Generator (MCG)

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

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

#### 6.9.2.6 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)         |

## Multipurpose Clock Generator (MCG)

# Chapter 7

## CMP: Analog Comparator Driver

### 7.1 Overview

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

The CMP driver is a basic comparator with advanced features. The APIs for the basic comparator enable the CMP as a general comparator, which compares two voltages of the two input channels and creates the output of the comparator result. The APIs for advanced features can be used as the plug-in function 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

```
int main(void)
{
    cmp_config_t mCmpConfigStruct;
    cmp_dac_config_t mCmpDacConfigStruct;

    // ...

    // Configures the comparator.
    CMP_Init(DEMO_CMP_INSTANCE);
    CMP_GetDefaultConfig(&mCmpConfigStruct);
    CMP_Configure(DEMO_CMP_INSTANCE, &mCmpConfigStruct);

    // Configures the DAC channel.
    mCmpDacConfigStruct.referenceVoltageSource =
        kCMP_VrefSourceVin2; // VCC.
    mCmpDacConfigStruct.DACValue = 32U; // Half voltage of logic high-level.
    CMP_SetDACConfig(DEMO_CMP_INSTANCE, &mCmpDacConfigStruct);
    CMP_SetInputChannels(DEMO_CMP_INSTANCE, DEMO_CMP_USER_CHANNEL, DEMO_CMP_DAC_CHANNEL
    );

    while (1)
    {
        if (0U != (kCMP_OutputAssertEventFlag &
        CMP_GetStatusFlags(DEMO_CMP_INSTANCE)))
        {
            // Do something.
        }
        else
        {
            // Do something.
        }
    }
}
```

#### 7.2.2 Interrupt Configuration

```
volatile uint32_t g_CmpFlags = 0U;
```

## Typical use case

```
// ...

void DEMO_CMP_IRQ_HANDLER_FUNC(void)
{
    g_CmpFlags = CMP_GetStatusFlags(DEMO_CMP_INSTANCE);
    CMP_ClearStatusFlags(DEMO_CMP_INSTANCE, kCMP_OutputRisingEventFlag |
        kCMP_OutputFallingEventFlag);
    if (0U != (g_CmpFlags & kCMP_OutputRisingEventFlag))
    {
        // Do something.
    }
    else if (0U != (g_CmpFlags & kCMP_OutputFallingEventFlag))
    {
        // Do something.
    }
}

int main(void)
{
    cmp_config_t mCmpConfigStruct;
    cmp_dac_config_t mCmpDacConfigStruct;

    // ...
    EnableIRQ(DEMO_CMP_IRQ_ID);
    // ...

    // Configures the comparator.
    CMP_Init(DEMO_CMP_INSTANCE);
    CMP_GetDefaultConfig(&mCmpConfigStruct);
    CMP_Configure(DEMO_CMP_INSTANCE, &mCmpConfigStruct);

    // Configures the DAC channel.
    mCmpDacConfigStruct.referenceVoltageSource =
        kCMP_VrefSourceVin2; // VCC.
    mCmpDacConfigStruct.DACValue = 32U; // Half voltage of logic high-level.
    CMP_SetDACConfig(DEMO_CMP_INSTANCE, &mCmpDacConfigStruct);
    CMP_SetInputChannels(DEMO_CMP_INSTANCE, DEMO_CMP_USER_CHANNEL, DEMO_CMP_DAC_CHANNEL
        );

    // Enables the output rising and falling interrupts.
    CMP_EnableInterrupts(DEMO_CMP_INSTANCE,
        kCMP_OutputRisingInterruptEnable |
        kCMP_OutputFallingInterruptEnable);

    while (1)
    {
    }
}
```

## Data Structures

- struct [cmp\\_config\\_t](#)  
*Configuration for the comparator.* [More...](#)
- struct [cmp\\_filter\\_config\\_t](#)  
*Configuration for the filter.* [More...](#)
- struct [cmp\\_dac\\_config\\_t](#)  
*Configuration for 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, 0)`)
- *CMP driver version 2.0.0.*

## Initialization

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

## Advanced Features

- void `CMP_SetFilterConfig` (`CMP_Type` \*base, const `cmp_filter_config_t` \*config)
   
    *Configures the filter.*
- void `CMP_SetDACConfig` (`CMP_Type` \*base, const `cmp_dac_config_t` \*config)
   
    *Configures the internal DAC.*
- void `CMP_EnableInterrupts` (`CMP_Type` \*base, `uint32_t` mask)
   
    *Enables the interrupts.*
- void `CMP_DisableInterrupts` (`CMP_Type` \*base, `uint32_t` mask)
   
    *Disables the interrupts.*

## Results

- `uint32_t` `CMP_GetStatusFlags` (`CMP_Type` \*base)

## Data Structure Documentation

- 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 comparison mode.*
- bool [enableInvertOutput](#)  
*Enable inverted comparator output.*
- bool [useUnfilteredOutput](#)  
*Set compare output(COUT) to equal COUTA(true) or COUT(false).*
- bool [enablePinOut](#)  
*The comparator output is available on the associated pin.*

##### 7.3.1.0.0.9 Field Documentation

###### 7.3.1.0.0.9.1 bool cmp\_config\_t::enableCmp

###### 7.3.1.0.0.9.2 cmp\_hysteresis\_mode\_t cmp\_config\_t::hysteresisMode

###### 7.3.1.0.0.9.3 bool cmp\_config\_t::enableHighSpeed

###### 7.3.1.0.0.9.4 bool cmp\_config\_t::enableInvertOutput

###### 7.3.1.0.0.9.5 bool cmp\_config\_t::useUnfilteredOutput

###### 7.3.1.0.0.9.6 bool cmp\_config\_t::enablePinOut

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

##### Data Fields

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

### 7.3.2.0.0.10 Field Documentation

#### 7.3.2.0.0.10.1 `uint8_t cmp_filter_config_t::filterCount`

Available range is 1-7, 0 would cause the filter disabled.

#### 7.3.2.0.0.10.2 `uint8_t cmp_filter_config_t::filterPeriod`

The divider to 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 DAC Output Voltage.*

### 7.3.3.0.0.11 Field Documentation

#### 7.3.3.0.0.11.1 `cmp_reference_voltage_source_t cmp_dac_config_t::referenceVoltageSource`

#### 7.3.3.0.0.11.2 `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, 0))`

## 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 compare output has occurred.

`kCMP_OutputFallingEventFlag` Falling-edge on compare output has occurred.

## Function Documentation

***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 resistor ladder network supply reference Vin.

***kCMP\_VrefSourceVin2*** Vin2 is selected as 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:

- Enabling the clock for CMP module.
- Configuring the comparator.
- Enabling the CMP module. Note: For some devices, multiple CMP instance share the same clock gate. In this case, to enable the clock for any instance enables all the CMPS. Check the chip reference manual for the clock assignment of the CMP.

Parameters

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

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

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

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

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

## Function Documentation

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> | Enable the module or not.    |

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

## Function Documentation

Parameters

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

### 7.6.10 `uint32_t CMP_GetStatusFlags ( CMP_Type * base )`

Parameters

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

Returns

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

### 7.6.11 `void CMP_ClearStatusFlags ( CMP_Type * base, uint32_t mask )`

Parameters

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

# Chapter 8

## CRC: Cyclic Redundancy Check Driver

### 8.1 Overview

The Kinetis SDK provides the Peripheral driver for the Cyclic Redundancy Check (CRC) module of Kinetis devices.

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

### 8.2 CRC Driver Initialization and Configuration

[CRC\\_Init\(\)](#) function enables the clock gate for the CRC module in the Kinetis SIM module and fully (re-)configures the CRC module according to configuration structure. The seed member of the configuration structure is the initial checksum for which new data can be added to. When starting new checksum computation, the seed shall be set to the initial checksum per the CRC protocol specification. For continued checksum operation, the seed shall be set to the intermediate checksum value as obtained from previous calls to [CRC\\_Get16bitResult\(\)](#) or [CRC\\_Get32bitResult\(\)](#) function. After [CRC\\_Init\(\)](#), one or multiple [CRC\\_WriteData\(\)](#) calls follow to update checksum with data, then [CRC\\_Get16bitResult\(\)](#) or [CRC\\_Get32bitResult\(\)](#) follows to read the result. The crcResult member of configuration structure determines if [CRC\\_Get16bitResult\(\)](#) or [CRC\\_Get32bitResult\(\)](#) return value is final checksum or intermediate checksum. [CRC\\_Init\(\)](#) can be called as many times as required, thus, allows for runtime changes of CRC protocol.

[CRC\\_GetDefaultConfig\(\)](#) function can be used to set the module configuration structure with parameters for CRC-16/CCIT-FALSE protocol.

### 8.3 CRC Write Data

The [CRC\\_WriteData\(\)](#) function is used to add data to actual CRC. Internally it tries to use 32-bit reads and writes for all aligned data in the user buffer and it uses 8-bit reads and writes for all unaligned data in the user buffer. This function can update CRC with user supplied data chunks of arbitrary size, so one can update CRC byte by byte or with all bytes at once. Prior call CRC configuration function [CRC\\_Init\(\)](#) fully specifies the CRC module configuration for [CRC\\_WriteData\(\)](#) call.

### 8.4 CRC Get Checksum

The [CRC\\_Get16bitResult\(\)](#) or [CRC\\_Get32bitResult\(\)](#) function is used to read the CRC module data register. Depending on prior CRC module usage the return value is either intermediate checksum or final checksum. Example: for 16-bit CRCs the following call sequences can be used:

[CRC\\_Init\(\)](#) / [CRC\\_WriteData\(\)](#) / [CRC\\_Get16bitResult\(\)](#) to get final checksum.

[CRC\\_Init\(\)](#) / [CRC\\_WriteData\(\)](#) / ... / [CRC\\_WriteData\(\)](#) / [CRC\\_Get16bitResult\(\)](#) to get final checksum.

## CRC Driver Examples

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

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

## 8.5 Comments about API usage in RTOS

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

The triplets

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

shall be protected by RTOS mutex to protect CRC module against concurrent accesses from different tasks. Example:

```
CRC_ModuleRTOS_Mutex_Lock;  
CRC_Init();  
CRC_WriteData();  
CRC_Get16bitResult();  
CRC_ModuleRTOS_Mutex_Unlock;
```

## 8.6 Comments about API usage in interrupt handler

All APIs can be used from interrupt handler although execution time shall be considered (interrupt latency of equal and lower priority interrupts increases). Protection against concurrent accesses from different interrupt handlers and/or tasks shall be assured by the user.

## 8.7 CRC Driver Examples

### 8.7.1 Simple examples

Simple example with default CRC-16/CCIT-FALSE protocol

```
crc_config_t config;  
CRC_Type *base;  
uint8_t data[] = {0x00, 0x01, 0x02, 0x03, 0x04};  
uint16_t checksum;  
  
base = CRC0;  
CRC_GetDefaultConfig(base, &config); /* default gives CRC-16/CCIT-FALSE */  
CRC_Init(base, &config);  
CRC_WriteData(base, data, sizeof(data));  
checksum = CRC_Get16bitResult(base);
```

Simple example with CRC-32 protocol configuration

```
crc_config_t config;  
uint32_t checksum;  
  
config.polynomial = 0x04C11DB7u;  
config.seed = 0xFFFFFFFFu;  
config.crcBits = kCrcBits32;  
config.reflectIn = true;
```

```

config.reflectOut = true;
config.complementChecksum = true;
config.crcResult = kCrcFinalChecksum;

CRC_Init(base, &config);
/* example: update by 1 byte at time */
while (dataSize)
{
    uint8_t c = GetCharacter();
    CRC_WriteData(base, &c, 1);
    dataSize--;
}
checksum = CRC_Get32bitResult(base);

```

## 8.7.2 Advanced examples

Per-partes data updates with context switch between. Assuming there are 3 tasks/threads, each using the CRC module to compute checksums of a different protocol, with context switches.

First, prepare three CRC module initialization functions for three different protocols: CRC-16 (ARC), CRC-16/CCIT-FALSE and CRC-32. Table below lists the individual protocol specifications. See also: <http://reveng.sourceforge.net/crc-catalogue/>

|                             | CRC-16/CCIT-FALSE | CRC-16  | CRC-32     |
|-----------------------------|-------------------|---------|------------|
| <b>Width</b>                | 16 bits           | 16 bits | 32 bits    |
| <b>Polynomial</b>           | 0x1021            | 0x8005  | 0x04C11DB7 |
| <b>Initial seed</b>         | 0xFFFF            | 0x0000  | 0xFFFFFFFF |
| <b>Complement check-sum</b> | No                | No      | Yes        |
| <b>Reflect In</b>           | No                | Yes     | Yes        |
| <b>Reflect Out</b>          | No                | Yes     | Yes        |

Corresponding init functions:

```

void InitCrc16_CCIT(CRC_Type *base, uint32_t seed, bool isLast)
{
    crc_config_t config;

    config.polynomial = 0x1021;
    config.seed = seed;
    config.reflectIn = false;
    config.reflectOut = false;
    config.complementChecksum = false;
    config.crcBits = kCrcBits16;
    config.crcResult = isLast?kCrcFinalChecksum:
        kCrcIntermediateChecksum;

    CRC_Init(base, &config);
}

void InitCrc16(CRC_Type *base, uint32_t seed, bool isLast)
{
    crc_config_t config;

```

## CRC Driver Examples

```
config.polynomial = 0x8005;
config.seed = seed;
config.reflectIn = true;
config.reflectOut = true;
config.complementChecksum = false;
config.crcBits = kCrcBits16;
config.crcResult = isLast?kCrcFinalChecksum:
    kCrcIntermediateChecksum;

    CRC_Init(base, &config);
}

void InitCrc32(CRC_Type *base, uint32_t seed, bool isLast)
{
    crc_config_t config;

    config.polynomial = 0x04C11DB7U;
    config.seed = seed;
    config.reflectIn = true;
    config.reflectOut = true;
    config.complementChecksum = true;
    config.crcBits = kCrcBits32;
    config.crcResult = isLast?kCrcFinalChecksum:
        kCrcIntermediateChecksum;

    CRC_Init(base, &config);
}
```

The following context switches show possible API usage:

```
uint16_t checksumCrc16;
uint32_t checksumCrc32;
uint16_t checksumCrc16Ccit;

checksumCrc16 = 0x0;
checksumCrc32 = 0xFFFFFFFFU;
checksumCrc16Ccit = 0xFFFFU;

/* Task A bytes[0-3] */
InitCrc16(base, checksumCrc16, false);
CRC_WriteData(base, &data[0], 4);
checksumCrc16 = CRC_Get16bitResult(base);

/* Task B bytes[0-3] */
InitCrc16_CCIT(base, checksumCrc16Ccit, false);
CRC_WriteData(base, &data[0], 4);
checksumCrc16Ccit = CRC_Get16bitResult(base);

/* Task C 4 bytes[0-3] */
InitCrc32(base, checksumCrc32, false);
CRC_WriteData(base, &data[0], 4);
checksumCrc32 = CRC_Get32bitResult(base);

/* Task D add final 5 bytes[4-8] */
InitCrc16_CCIT(base, checksumCrc16Ccit, true);
CRC_WriteData(base, &data[4], 5);
checksumCrc16Ccit = CRC_Get16bitResult(base);

/* Task E 3 bytes[4-6] */
InitCrc32(base, checksumCrc32, false);
CRC_WriteData(base, &data[4], 3);
checksumCrc32 = CRC_Get32bitResult(base);

/* Task F 3 bytes[4-6] */
InitCrc16(base, checksumCrc16, false);
```

```

CRC_WriteData(base, &data[4], 3);
checksumCrc16 = CRC_Get16bitResult(base);

/* Task C add final 2 bytes[7-8] */
InitCrc32(base, checksumCrc32, true);
CRC_WriteData(base, &data[7], 2);
checksumCrc32 = CRC_Get32bitResult(base);

/* Task A add final 2 bytes[7-8] */
InitCrc16(base, checksumCrc16, true);
CRC_WriteData(base, &data[7], 2);
checksumCrc16 = CRC_Get16bitResult(base);

```

## Data Structures

- struct **crc\_config\_t**  
*CRC protocol configuration.* [More...](#)

## Macros

- #define **CRC\_DRIVER\_USE\_CRC16\_CCIT\_FALSE\_AS\_DEFAULT** 1  
*Default configuration structure filled by [CRC\\_GetDefaultConfig\(\)](#).*

## Enumerations

- enum **crc\_bits\_t** {
 kCrcBits16 = 0U,
 kCrcBits32 = 1U }
   
*CRC bit width.*
- enum **crc\_result\_t** {
 kCrcFinalChecksum = 0U,
 kCrcIntermediateChecksum = 1U }
   
*CRC result type.*

## Functions

- void **CRC\_Init** (CRC\_Type \*base, const **crc\_config\_t** \*config)  
*Enables and configures the CRC peripheral module.*
- static void **CRC\_Deinit** (CRC\_Type \*base)  
*Disables the CRC peripheral module.*
- void **CRC\_GetDefaultConfig** (**crc\_config\_t** \*config)  
*Loads default values to CRC protocol configuration structure.*
- void **CRC\_WriteData** (CRC\_Type \*base, const uint8\_t \*data, size\_t dataSize)  
*Writes data to the CRC module.*
- uint32\_t **CRC\_Get32bitResult** (CRC\_Type \*base)  
*Reads 32-bit checksum from the CRC module.*
- uint16\_t **CRC\_Get16bitResult** (CRC\_Type \*base)  
*Reads 16-bit checksum from the CRC module.*

## Driver version

- #define **FSL\_CRC\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 1))  
*CRC driver version.*

## Macro Definition Documentation

### 8.8 Data Structure Documentation

#### 8.8.1 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\(\)](#)*

#### 8.8.1.0.0.12 Field Documentation

##### 8.8.1.0.0.12.1 uint32\_t crc\_config\_t::polynomial

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

##### 8.8.1.0.0.12.2 bool crc\_config\_t::reflectIn

##### 8.8.1.0.0.12.3 bool crc\_config\_t::reflectOut

##### 8.8.1.0.0.12.4 bool crc\_config\_t::complementChecksum

##### 8.8.1.0.0.12.5 crc\_bits\_t crc\_config\_t::crcBits

### 8.9 Macro Definition Documentation

#### 8.9.1 #define FSL\_CRC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

Version 2.0.1.

Current version: 2.0.1

Change log:

- Version 2.0.1
  - move DATA and DATALL macro definition from header file to source file

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

Use CRC16-CCIT-FALSE as default.

## 8.10 Enumeration Type Documentation

### 8.10.1 enum crc\_bits\_t

Enumerator

*kCrcBits16* Generate 16-bit CRC code.

*kCrcBits32* Generate 32-bit CRC code.

### 8.10.2 enum crc\_result\_t

Enumerator

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

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

## 8.11 Function Documentation

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

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

Parameters

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

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

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

## Function Documentation

Parameters

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

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

Loads default values to CRC protocol configuration structure. The default values are:

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

Parameters

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

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

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

Parameters

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

### 8.11.5 **uint32\_t** CRC\_Get32bitResult ( **CRC\_Type** \* *base* )

Reads CRC data register (intermediate or final checksum). The configured type of transpose and complement are applied.

Parameters

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

Returns

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

### **8.11.6 uint16\_t CRC\_Get16bitResult ( CRC\_Type \* *base* )**

Reads CRC data register (intermediate or final checksum). The configured type of transpose and complement are applied.

Parameters

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

Returns

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

## Function Documentation

# Chapter 9

## DAC: Digital-to-Analog Converter Driver

### 9.1 Overview

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

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

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

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

### 9.2 Typical use case

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

```
// ...  
  
// Configures the DAC.  
DAC_GetDefaultConfig(&dacConfigStruct);  
DAC_Init(DEMO_DAC_INSTANCE, &dacConfigStruct);  
DAC_Enable(DEMO_DAC_INSTANCE, true);  
DAC_SetBufferReadPointer(DEMO_DAC_INSTANCE, 0U);  
  
// ...  
  
DAC_SetBufferValue(DEMO_DAC_INSTANCE, 0U, dacValue);
```

#### 9.2.2 Working with the hardware buffer.

```
// ...  
  
EnableIRQ(DEMO_DAC IRQ_ID);  
  
// ...  
  
// Configures the DAC.  
DAC_GetDefaultConfig(&dacConfigStruct);  
DAC_Init(DEMO_DAC_INSTANCE, &dacConfigStruct);  
DAC_Enable(DEMO_DAC_INSTANCE, true);  
// Configures the DAC buffer.  
DAC_GetDefaultBufferConfig(&dacBufferConfigStruct);
```

## Typical use case

```
DAC_SetBufferConfig(DEMO_DAC_INSTANCE, &dacBufferConfigStruct);
DAC_SetBufferReadPointer(DEMO_DAC_INSTANCE, 0U); // Make sure the read pointer
      to the start.
for (index = 0U, dacValue = 0; index < DEMO_DAC_USED_BUFFER_SIZE; index++, dacValue += (0xFFFFU /
    DEMO_DAC_USED_BUFFER_SIZE))
{
    DAC_SetBufferValue(DEMO_DAC_INSTANCE, index, dacValue);
}
// Clears flags.
#if defined(FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION) && FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION
    g_DacBufferWatermarkInterruptFlag = false;
#endif // FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION
    g_DacBufferReadPointerTopPositionInterruptFlag = false;
    g_DacBufferReadPointerBottomPositionInterruptFlag = false;

// Enables interrupts.
mask = 0U;
#if defined(FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION) && FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION
    mask |= kDAC_BufferWatermarkInterruptEnable;
#endif // FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION
    mask |= kDAC_BufferReadPointerTopInterruptEnable |
        kDAC_BufferReadPointerBottomInterruptEnable;
    DAC_EnableBuffer(DEMO_DAC_INSTANCE, true);
    DAC_EnableBufferInterrupts(DEMO_DAC_INSTANCE, mask);

// ISR for the DAC interrupt.
void DEMO_DAC_IRQ_HANDLER_FUNC(void)
{
    uint32_t flags = DAC_GetBufferStatusFlags(DEMO_DAC_INSTANCE);

#if defined(FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION) && FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION
    if (kDAC_BufferWatermarkFlag == (kDAC_BufferWatermarkFlag & flags))
    {
        g_DacBufferWatermarkInterruptFlag = true;
    }
#endif // FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION
    if (kDAC_BufferReadPointerTopPositionFlag == (
        kDAC_BufferReadPointerTopPositionFlag & flags))
    {
        g_DacBufferReadPointerTopPositionInterruptFlag = true;
    }
    if (kDAC_BufferReadPointerBottomPositionFlag == (
        kDAC_BufferReadPointerBottomPositionFlag & flags))
    {
        g_DacBufferReadPointerBottomPositionInterruptFlag = true;
    }
    DAC_ClearBufferStatusFlags(DEMO_DAC_INSTANCE, flags); /* Clear flags. */
}
```

## Data Structures

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

## Enumerations

- enum `_dac_buffer_status_flags` {  
    `kDAC_BufferReadPointerTopPositionFlag` = `DAC_SR_DACBFRPTF_MASK`,  
    `kDAC_BufferReadPointerBottomPositionFlag` = `DAC_SR_DACBFRPBF_MASK` }  
*DAC buffer flags.*

- enum `_dac_buffer_interrupt_enable` {
   
  `kDAC_BufferReadPointerTopInterruptEnable` = DAC\_C0\_DACBTIEN\_MASK,
   
  `kDAC_BufferReadPointerBottomInterruptEnable` = DAC\_C0\_DACBBIEN\_MASK }

*DAC buffer interrupts.*

- enum `dac_reference_voltage_source_t` {
   
  `kDAC_ReferenceVoltageSourceVref1` = 0U,
   
  `kDAC_ReferenceVoltageSourceVref2` = 1U }

*DAC reference voltage source.*

- enum `dac_buffer_trigger_mode_t` {
   
  `kDAC_BufferTriggerByHardwareMode` = 0U,
   
  `kDAC_BufferTriggerBySoftwareMode` = 1U }

*DAC buffer trigger mode.*

- enum `dac_buffer_work_mode_t` {
   
  `kDAC_BufferWorkAsNormalMode` = 0U,
   
  `kDAC_BufferWorkAsOneTimeScanMode` }

*DAC buffer work mode.*

## Driver version

- #define `FSL_DAC_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 1))
- DAC driver version 2.0.1.*

## Initialization

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

## Buffer

- static void `DAC_EnableBuffer` (DAC\_Type \*base, bool enable)
   
    *Enables the DAC buffer.*
- void `DAC_SetBufferConfig` (DAC\_Type \*base, const `dac_buffer_config_t` \*config)
   
    *Configures the CMP buffer.*
- void `DAC_GetDefaultBufferConfig` (`dac_buffer_config_t` \*config)
   
    *Initializes the DAC buffer configuration structure.*
- static void `DAC_EnableBufferDMA` (DAC\_Type \*base, bool enable)
   
    *Enables the DMA for DAC buffer.*
- void `DAC_SetBufferValue` (DAC\_Type \*base, uint8\_t index, uint16\_t value)
   
    *Sets the value for items in the buffer.*
- static void `DAC_DoSoftwareTriggerBuffer` (DAC\_Type \*base)
   
    *Triggers the buffer by software and updates the read pointer of the DAC buffer.*
- static uint8\_t `DAC_GetBufferReadPointer` (DAC\_Type \*base)
   
    *Gets the current read pointer of the DAC buffer.*
- void `DAC_SetBufferReadPointer` (DAC\_Type \*base, uint8\_t index)

## Data Structure Documentation

- void [DAC\\_EnableBufferInterrupts](#) (DAC\_Type \*base, uint32\_t mask)  
*Enables interrupts for the DAC buffer.*
- void [DAC\\_DisableBufferInterrupts](#) (DAC\_Type \*base, uint32\_t mask)  
*Disables interrupts for the DAC buffer.*
- uint32\_t [DAC\\_GetBufferStatusFlags](#) (DAC\_Type \*base)  
*Gets the flags of events for the DAC buffer.*
- void [DAC\\_ClearBufferStatusFlags](#) (DAC\_Type \*base, uint32\_t mask)  
*Clears the flags of events for the DAC buffer.*

### 9.3 Data Structure Documentation

#### 9.3.1 struct dac\_config\_t

##### Data Fields

- [dac\\_reference\\_voltage\\_source\\_t referenceVoltageSource](#)  
*Select the DAC reference voltage source.*
- bool [enableLowPowerMode](#)  
*Enable the low-power mode.*

##### 9.3.1.0.0.13 Field Documentation

###### 9.3.1.0.0.13.1 [dac\\_reference\\_voltage\\_source\\_t dac\\_config\\_t::referenceVoltageSource](#)

###### 9.3.1.0.0.13.2 [bool dac\\_config\\_t::enableLowPowerMode](#)

#### 9.3.2 struct dac\_buffer\_config\_t

##### Data Fields

- [dac\\_buffer\\_trigger\\_mode\\_t triggerMode](#)  
*Select the buffer's trigger mode.*
- [dac\\_buffer\\_work\\_mode\\_t workMode](#)  
*Select the buffer's work mode.*
- uint8\_t [upperLimit](#)  
*Set the upper limit for buffer index.*

##### 9.3.2.0.0.14 Field Documentation

###### 9.3.2.0.0.14.1 [dac\\_buffer\\_trigger\\_mode\\_t dac\\_buffer\\_config\\_t::triggerMode](#)

###### 9.3.2.0.0.14.2 [dac\\_buffer\\_work\\_mode\\_t dac\\_buffer\\_config\\_t::workMode](#)

###### 9.3.2.0.0.14.3 [uint8\\_t dac\\_buffer\\_config\\_t::upperLimit](#)

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

## 9.4 Macro Definition Documentation

### 9.4.1 #define FSL\_DAC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

## 9.5 Enumeration Type Documentation

### 9.5.1 enum \_dac\_buffer\_status\_flags

Enumerator

*kDAC\_BufferReadPointerTopPositionFlag* DAC Buffer Read Pointer Top Position Flag.

*kDAC\_BufferReadPointerBottomPositionFlag* DAC Buffer Read Pointer Bottom Position Flag.

### 9.5.2 enum \_dac\_buffer\_interrupt\_enable

Enumerator

*kDAC\_BufferReadPointerTopInterruptEnable* DAC Buffer Read Pointer Top Flag Interrupt Enable.

*kDAC\_BufferReadPointerBottomInterruptEnable* DAC Buffer Read Pointer Bottom Flag Interrupt Enable.

### 9.5.3 enum dac\_reference\_voltage\_source\_t

Enumerator

*kDAC\_ReferenceVoltageSourceVref1* The DAC selects DACREF\_1 as the reference voltage.

*kDAC\_ReferenceVoltageSourceVref2* The DAC selects DACREF\_2 as the reference voltage.

### 9.5.4 enum dac\_buffer\_trigger\_mode\_t

Enumerator

*kDAC\_BufferTriggerByHardwareMode* The DAC hardware trigger is selected.

*kDAC\_BufferTriggerBySoftwareMode* The DAC software trigger is selected.

### 9.5.5 enum dac\_buffer\_work\_mode\_t

Enumerator

*kDAC\_BufferWorkAsNormalMode* Normal mode.

*kDAC\_BufferWorkAsOneTimeScanMode* One-Time Scan mode.

## Function Documentation

### 9.6 Function Documentation

#### 9.6.1 void DAC\_Init ( **DAC\_Type** \* *base*, **const dac\_config\_t** \* *config* )

This function initializes the DAC module, including:

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

Parameters

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

#### 9.6.2 void DAC\_Deinit ( **DAC\_Type** \* *base* )

This function de-initializes the DAC module, including:

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

Parameters

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

#### 9.6.3 void DAC\_GetDefaultConfig ( **dac\_config\_t** \* *config* )

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

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

Parameters

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

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

Parameters

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

### 9.6.5 static void DAC\_EnableBuffer ( DAC\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

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

### 9.6.6 void DAC\_SetBufferConfig ( DAC\_Type \* *base*, const dac\_buffer\_config\_t \* *config* )

Parameters

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

### 9.6.7 void DAC\_GetDefaultBufferConfig ( dac\_buffer\_config\_t \* *config* )

This function initializes the DAC buffer configuration structure to a default value. The default values are:

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

Parameters

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

### 9.6.8 static void DAC\_EnableBufferDMA ( DAC\_Type \* *base*, bool *enable* ) [inline], [static]

## Function Documentation

Parameters

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

### **9.6.9 void DAC\_SetBufferValue ( DAC\_Type \* *base*, uint8\_t *index*, uint16\_t *value* )**

Parameters

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

### **9.6.10 static void DAC\_DoSoftwareTriggerBuffer ( DAC\_Type \* *base* ) [inline], [static]**

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

Parameters

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

### **9.6.11 static uint8\_t DAC\_GetBufferReadPointer ( DAC\_Type \* *base* ) [inline], [static]**

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

Parameters

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

Returns

Current read pointer of DAC buffer.

**9.6.12 void DAC\_SetBufferReadPointer ( DAC\_Type \* *base*, uint8\_t *index* )**

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

Parameters

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

**9.6.13 void DAC\_EnableBufferInterrupts ( DAC\_Type \* *base*, uint32\_t *mask* )**

Parameters

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

**9.6.14 void DAC\_DisableBufferInterrupts ( DAC\_Type \* *base*, uint32\_t *mask* )**

Parameters

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

**9.6.15 uint32\_t DAC\_GetBufferStatusFlags ( DAC\_Type \* *base* )**

Parameters

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

Returns

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

**9.6.16 void DAC\_ClearBufferStatusFlags ( DAC\_Type \* *base*, uint32\_t *mask* )**

## Function Documentation

### Parameters

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

# Chapter 10

## DMAMUX: Direct Memory Access Multiplexer Driver

### 10.1 Overview

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

### 10.2 Typical use case

#### 10.2.1 DMAMUX Operation

```
DMAMUX_Init(DMAMUX0);
DMAMUX_SetSource(DMAMUX0, channel, source);
DMAMUX_EnableChannel(DMAMUX0, channel);
...
DMAMUX_DisableChannel(DMAMUX, channel);
DMAMUX_Deinit(DMAMUX0);
```

#### Driver version

- #define **FSL\_DMAMUX\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 1))  
*DMAMUX driver version 2.0.1.*

#### DMAMUX Initialize and De-initialize

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

#### DMAMUX Channel Operation

- static void **DMAMUX\_EnableChannel** (DMAMUX\_Type \*base, uint32\_t channel)  
*Enable DMAMUX channel.*
- static void **DMAMUX\_DisableChannel** (DMAMUX\_Type \*base, uint32\_t channel)  
*Disable DMAMUX channel.*
- static void **DMAMUX\_SetSource** (DMAMUX\_Type \*base, uint32\_t channel, uint32\_t source)  
*Configure DMAMUX channel source.*

### 10.3 Macro Definition Documentation

#### 10.3.1 #define **FSL\_DMAMUX\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 1))

## Function Documentation

### 10.4 Function Documentation

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

This function ungates the DMAMUX clock.

Parameters

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

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

This function gate the DMAMUX clock.

Parameters

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

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

This function enable DMAMUX channel to work.

Parameters

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

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

This function disable DMAMUX channel.

Note

User must disable DMAMUX channel before configuring it.

Parameters

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

## Function Documentation

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

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

#### Parameters

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

# Chapter 11

## DSPI: Serial Peripheral Interface Driver

### 11.1 Overview

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

### Modules

- DSPI DMA Driver
- DSPI Driver
- DSPI FreeRTOS Driver
- DSPI eDMA Driver
- DSPI µCOS/II Driver
- DSPI µCOS/III Driver

## DSPI Driver

### 11.2 DSPI Driver

#### 11.2.1 Overview

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

#### 11.2.2 Typical use case

##### 11.2.2.1 Master Operation

```
dspi_master_handle_t g_m_handle; //global variable
dspi_master_config_t masterConfig;
masterConfig.whichCtar = kDSPI_Ctar0;
masterConfig.ctarConfig.baudRate = baudrate;
masterConfig.ctarConfig.bitsPerFrame = 8;
masterConfig.ctarConfig.cpol = kDSPI_ClockPolarityActiveHigh;
masterConfig.ctarConfig.cpha = kDSPI_ClockPhaseFirstEdge;
masterConfig.ctarConfig.direction = kDSPI_MsbFirst;
masterConfig.ctarConfig.pcsToSckDelayInNanoSec = 1000000000 / baudrate;
masterConfig.ctarConfig.lastSckToPcsDelayInNanoSec = 1000000000 / baudrate;
masterConfig.ctarConfig.betweenTransferDelayInNanoSec = 1000000000 / baudrate;
masterConfig.whichPcs = kDSPI_Pcs0;
masterConfig.pcsActiveHighOrLow = kDSPI_PcsActiveLow;
masterConfig.enableContinuousSCK = false;
masterConfig.enableRx_fifoOverWrite = false;
masterConfig.enableModifiedTimingFormat = false;
masterConfig.samplePoint = kDSPI_SckToSin0Clock;
DSPI_MasterInit(base, &masterConfig, srcClock_Hz);

//srcClock_Hz = CLOCK_GetFreq(xxx);
DSPI_MasterInit(base, &masterConfig, srcClock_Hz);

DSPI_MasterTransferCreateHandle(base, &g_m_handle, NULL, NULL);

masterXfer.txData = masterSendBuffer;
masterXfer.rxData = masterReceiveBuffer;
masterXfer.dataSize = transfer_dataSize;
masterXfer.configFlags = kDSPI_MasterCtar0 | kDSPI_MasterPcs0 ;
DSPI_MasterTransferBlocking(base, &g_m_handle, &masterXfer);
```

##### 11.2.2.2 Slave Operation

```
dspi_slave_handle_t g_s_handle;//global variable
/*Slave config*/
slaveConfig.whichCtar = kDSPI_Ctar0;
slaveConfig.ctarConfig.bitsPerFrame = 8;
slaveConfig.ctarConfig.cpol = kDSPI_ClockPolarityActiveHigh;
slaveConfig.ctarConfig.cpha = kDSPI_ClockPhaseFirstEdge;
```

```

slaveConfig.enableContinuousSCK      = false;
slaveConfig.enableRxFifoOverWrite   = false;
slaveConfig.enableModifiedTimingFormat = false;
slaveConfig.samplePoint             = kDSPI_SckToSin0Clock;
DSPI_SlaveInit(base, &slaveConfig);

slaveXfer.txData      = slaveSendBuffer0;
slaveXfer.rxData      = slaveReceiveBuffer0;
slaveXfer.dataSize    = transfer_dataSize;
slaveXfer.configFlags = kDSPI_SlaveCtar0;

bool isTransferCompleted = false;
DSPI_SlaveTransferCreateHandle(base, &g_s_handle, DSPI_SlaveUserCallback, &
                               isTransferCompleted);

DSPI_SlaveTransferNonBlocking(&g_s_handle, &slaveXfer);

//void DSPI_SlaveUserCallback(SPI_Type *base, dspi_slave_handle_t *handle, status_t status, void
//                             *isTransferCompleted)
//{
//    if (status == kStatus_Success)
//    {
//        __NOP();
//    }
//    else if (status == kStatus_DSPI_Error)
//    {
//        __NOP();
//    }
//    *(bool *)isTransferCompleted = true;
//    PRINTF("This is DSPI slave call back . \r\n");
//}

```

## Data Structures

- struct [dsPIC33F DSPI command data configuration structure](#)  
*DSPI master command date configuration used for SPIx\_PUSHR.* [More...](#)
- struct [dsPIC33F DSPI master ctar configuration structure](#)  
*DSPI master ctar configuration structure.* [More...](#)
- struct [dsPIC33F DSPI master configuration structure](#)  
*DSPI master configuration structure.* [More...](#)
- struct [dsPIC33F DSPI slave ctar configuration structure](#)  
*DSPI slave ctar configuration structure.* [More...](#)
- struct [dsPIC33F DSPI slave configuration structure](#)  
*DSPI slave configuration structure.* [More...](#)
- struct [dsPIC33F DSPI transfer structure](#)  
*DSPI master/slave transfer structure.* [More...](#)
- struct [dsPIC33F DSPI master handle structure](#)  
*DSPI master transfer handle structure used for transactional API.* [More...](#)
- struct [dsPIC33F DSPI slave handle structure](#)  
*DSPI slave transfer handle structure used for transactional API.* [More...](#)

## Macros

- #define [DSPI\\_DUMMY\\_DATA](#) (0x00U)

## DSPI Driver

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

## Typedefs

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

## Enumerations

- `enum _dspi_status {`  
    `kStatus_DSPI_Busy = MAKE_STATUS(kStatusGroup_DSPI, 0),`  
    `kStatus_DSPI_Error = MAKE_STATUS(kStatusGroup_DSPI, 1),`  
    `kStatus_DSPI_Idle = MAKE_STATUS(kStatusGroup_DSPI, 2),`  
    `kStatus_DSPI_OutOfRange = MAKE_STATUS(kStatusGroup_DSPI, 3) }`  
*Status for the DSPI driver.*
- `enum _dspi_flags {`  
    `kDSPI_TxCompleteFlag = SPI_SR_TCF_MASK,`  
    `kDSPI_EndOfQueueFlag = SPI_SR_EOQF_MASK,`  
    `kDSPI_TxFifoUnderflowFlag = SPI_SR_TFUF_MASK,`  
    `kDSPI_TxFifoFillRequestFlag = SPI_SR_TFFF_MASK,`  
    `kDSPI_RxFifoOverflowFlag = SPI_SR_RFOF_MASK,`  
    `kDSPI_RxFifoDrainRequestFlag = SPI_SR_RFDF_MASK,`  
    `kDSPI_TxAndRxStatusFlag = SPI_SR_TXRXS_MASK,`  
    `kDSPI_AllStatusFlag }`  
*DSPI status flags in SPIx\_SR register.*
- `enum _dspi_interrupt_enable {`

- ```

kDSPI_TxCompleteInterruptEnable = SPI_RSER_TCF_RE_MASK,
kDSPI_EndOfQueueInterruptEnable = SPI_RSER_EOQF_RE_MASK,
kDSPI_TxFifoUnderflowInterruptEnable = SPI_RSER_TFUF_RE_MASK,
kDSPI_TxFifoFillRequestInterruptEnable = SPI_RSER_TFFF_RE_MASK,
kDSPI_RxFifoOverflowInterruptEnable = SPI_RSER_RFOF_RE_MASK,
kDSPI_RxFifoDrainRequestInterruptEnable = SPI_RSER_RFDF_RE_MASK,
kDSPI_AllInterruptEnable }

DSPI interrupt source.
• enum _dspi_dma_enable {
  kDSPI_TxDmaEnable = (SPI_RSER_TFFF_RE_MASK | SPI_RSER_TFFF_DIRS_MASK),
  kDSPI_RxDmaEnable = (SPI_RSER_RFDF_RE_MASK | SPI_RSER_RFDF_DIRS_MASK) }

DSPI DMA source.
• enum dspi_master_slave_mode_t {
  kDSPI_Master = 1U,
  kDSPI_Slave = 0U }

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

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

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

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

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

DSPI clock polarity configuration for a given CTAR.
• enum dspi_clock_phase_t {

```

## DSPI Driver

- ```
kDSPI_ClockPhaseFirstEdge = 0U,  
kDSPI_ClockPhaseSecondEdge = 1U }  
    DSPI clock phase configuration for a given CTAR.  
• enum dspi_shift_direction_t {  
    kDSPI_MsbFirst = 0U,  
    kDSPI_LsbFirst = 1U }  
    DSPI data shifter direction options for a given CTAR.  
• enum dspi_delay_type_t {  
    kDSPI_PcsToSck = 1U,  
    kDSPI_LastSckToPcs,  
    kDSPI_BetweenTransfer }  
    DSPI delay type selection.  
• enum dspi_ctar_selection_t {  
    kDSPI_Ctar0 = 0U,  
    kDSPI_Ctar1 = 1U,  
    kDSPI_Ctar2 = 2U,  
    kDSPI_Ctar3 = 3U,  
    kDSPI_Ctar4 = 4U,  
    kDSPI_Ctar5 = 5U,  
    kDSPI_Ctar6 = 6U,  
    kDSPI_Ctar7 = 7U }  
    DSPI Clock and Transfer Attributes Register (CTAR) selection.  
• enum _dspi_transfer_config_flag_for_master {  
    kDSPI_MasterCtar0 = 0U << DSPI_MASTER_CTAR_SHIFT,  
    kDSPI_MasterCtar1 = 1U << DSPI_MASTER_CTAR_SHIFT,  
    kDSPI_MasterCtar2 = 2U << DSPI_MASTER_CTAR_SHIFT,  
    kDSPI_MasterCtar3 = 3U << DSPI_MASTER_CTAR_SHIFT,  
    kDSPI_MasterCtar4 = 4U << DSPI_MASTER_CTAR_SHIFT,  
    kDSPI_MasterCtar5 = 5U << DSPI_MASTER_CTAR_SHIFT,  
    kDSPI_MasterCtar6 = 6U << DSPI_MASTER_CTAR_SHIFT,  
    kDSPI_MasterCtar7 = 7U << DSPI_MASTER_CTAR_SHIFT,  
    kDSPI_MasterPcs0 = 0U << DSPI_MASTER_PCS_SHIFT,  
    kDSPI_MasterPcs1 = 1U << DSPI_MASTER_PCS_SHIFT,  
    kDSPI_MasterPcs2 = 2U << DSPI_MASTER_PCS_SHIFT,  
    kDSPI_MasterPcs3 = 3U << DSPI_MASTER_PCS_SHIFT,  
    kDSPI_MasterPcs4 = 4U << DSPI_MASTER_PCS_SHIFT,  
    kDSPI_MasterPcs5 = 5U << DSPI_MASTER_PCS_SHIFT,  
    kDSPI_MasterPcsContinuous = 1U << 20,  
    kDSPI_MasterActiveAfterTransfer = 1U << 21 }  
    Can use this enumeration for DSPI master transfer configFlags.  
• enum _dspi_transfer_config_flag_for_slave { kDSPI_SlaveCtar0 = 0U << DSPI_SLAVE_CTAR_SHIFT }  
    Can use this enum for DSPI slave transfer configFlags.  
• enum _dspi_transfer_state {  
    kDSPI_Idle = 0x0U,  
    kDSPI_Busy,
```

```
kDSPI_Error }
```

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

## Driver version

- #define **FSL\_DSPI\_DRIVER\_VERSION** (MAKE\_VERSION(2, 1, 1))  
*DSPI driver version 2.1.1.*

## Initialization and deinitialization

- void **DSPI\_MasterInit** (SPI\_Type \*base, const **dspi\_master\_config\_t** \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes the DSPI master.*
- void **DSPI\_MasterGetDefaultConfig** (**dspi\_master\_config\_t** \*masterConfig)  
*Sets the **dspi\_master\_config\_t** structure to default values.*
- void **DSPI\_SlaveInit** (SPI\_Type \*base, const **dspi\_slave\_config\_t** \*slaveConfig)  
*DSPI slave configuration.*
- void **DSPI\_SlaveGetDefaultConfig** (**dspi\_slave\_config\_t** \*slaveConfig)  
*Sets the **dspi\_slave\_config\_t** structure to default values.*
- void **DSPI\_Deinit** (SPI\_Type \*base)  
*De-initializes the DSPI peripheral.*
- static void **DSPI\_Enable** (SPI\_Type \*base, bool enable)  
*Enables the DSPI peripheral and sets the MCR MDIS to 0.*

## Status

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

## Interrupts

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

## DMA Control

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

## DSPI Driver

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

## Bus Operations

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

- static uint32\_t **DSPI\_ReadData** (SPI\_Type \*base)  
*Reads data from the data buffer.*

## Transactional

- void **DSPI\_MasterTransferCreateHandle** (SPI\_Type \*base, dspi\_master\_handle\_t \*handle, **dspi\_master\_transfer\_callback\_t** callback, void \*userData)  
*Initializes the DSPI master handle.*
- status\_t **DSPI\_MasterTransferBlocking** (SPI\_Type \*base, **dspi\_transfer\_t** \*transfer)  
*DSPI master transfer data using polling.*
- status\_t **DSPI\_MasterTransferNonBlocking** (SPI\_Type \*base, dspi\_master\_handle\_t \*handle, **dspi\_transfer\_t** \*transfer)  
*DSPI master transfer data using interrupts.*
- status\_t **DSPI\_MasterTransferGetCount** (SPI\_Type \*base, dspi\_master\_handle\_t \*handle, size\_t \*count)  
*Gets the master transfer count.*
- void **DSPI\_MasterTransferAbort** (SPI\_Type \*base, dspi\_master\_handle\_t \*handle)  
*DSPI master aborts transfer using an interrupt.*
- void **DSPI\_MasterTransferHandleIRQ** (SPI\_Type \*base, dspi\_master\_handle\_t \*handle)  
*DSPI Master IRQ handler function.*
- void **DSPI\_SlaveTransferCreateHandle** (SPI\_Type \*base, dspi\_slave\_handle\_t \*handle, **dspi\_slave\_transfer\_callback\_t** callback, void \*userData)  
*Initializes the DSPI slave handle.*
- status\_t **DSPI\_SlaveTransferNonBlocking** (SPI\_Type \*base, dspi\_slave\_handle\_t \*handle, **dspi\_transfer\_t** \*transfer)  
*DSPI slave transfers data using an interrupt.*
- status\_t **DSPI\_SlaveTransferGetCount** (SPI\_Type \*base, dspi\_slave\_handle\_t \*handle, size\_t \*count)  
*Gets the slave transfer count.*
- void **DSPI\_SlaveTransferAbort** (SPI\_Type \*base, dspi\_slave\_handle\_t \*handle)  
*DSPI slave aborts a transfer using an interrupt.*
- void **DSPI\_SlaveTransferHandleIRQ** (SPI\_Type \*base, dspi\_slave\_handle\_t \*handle)  
*DSPI Master IRQ handler function.*

### 11.2.3 Data Structure Documentation

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

##### Data Fields

- bool **isPcsContinuous**  
*Option to enable the continuous assertion of chip select between transfers.*
- **dspi\_ctar\_selection\_t whichCtar**  
*The desired Clock and Transfer Attributes Register (CTAR) to use for CTAS.*
- **dspi\_which\_pcs\_t whichPcs**  
*The desired PCS signal to use for the data transfer.*
- bool **isEndOfQueue**

## DSPI Driver

- **bool clearTransferCount**  
*Signals that the current transfer is the last in the queue.*  
*Clears SPI Transfer Counter (SPI\_TCNT) before transmission starts.*

### 11.2.3.1.0.15 Field Documentation

**11.2.3.1.0.15.1 bool dspi\_command\_data\_config\_t::isPcsContinuous**

**11.2.3.1.0.15.2 dspi\_ctar\_selection\_t dspi\_command\_data\_config\_t::whichCtar**

**11.2.3.1.0.15.3 dspi\_which\_pcs\_t dspi\_command\_data\_config\_t::whichPcs**

**11.2.3.1.0.15.4 bool dspi\_command\_data\_config\_t::isEndOfQueue**

**11.2.3.1.0.15.5 bool dspi\_command\_data\_config\_t::clearTransferCount**

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

#### Data Fields

- **uint32\_t baudRate**  
*Baud Rate for DSPI.*
- **uint32\_t bitsPerFrame**  
*Bits per frame, minimum 4, maximum 16.*
- **dspi\_clock\_polarity\_t cpol**  
*Clock polarity.*
- **dspi\_clock\_phase\_t cpha**  
*Clock phase.*
- **dspi\_shift\_direction\_t direction**  
*MSB or LSB data shift direction.*
- **uint32\_t pcsToSckDelayInNanoSec**  
*PCS to SCK delay time with nanosecond , set to 0 sets the minimum delay.*
- **uint32\_t lastSckToPcsDelayInNanoSec**  
*Last SCK to PCS delay time with nanosecond , set to 0 sets the minimum delay. It sets the boundary value if out of range that can be set.*
- **uint32\_t betweenTransferDelayInNanoSec**  
*After SCK delay time with nanosecond , set to 0 sets the minimum delay. It sets the boundary value if out of range that can be set.*

### 11.2.3.2.0.16 Field Documentation

**11.2.3.2.0.16.1 uint32\_t dspi\_master\_ctar\_config\_t::baudRate**

**11.2.3.2.0.16.2 uint32\_t dspi\_master\_ctar\_config\_t::bitsPerFrame**

**11.2.3.2.0.16.3 dspi\_clock\_polarity\_t dspi\_master\_ctar\_config\_t::cpol**

**11.2.3.2.0.16.4 dspi\_clock\_phase\_t dspi\_master\_ctar\_config\_t::cpha**

**11.2.3.2.0.16.5 dspi\_shift\_direction\_t dspi\_master\_ctar\_config\_t::direction**

**11.2.3.2.0.16.6 uint32\_t dspi\_master\_ctar\_config\_t::pcsToSckDelayInNanoSec**

It sets the boundary value if out of range that can be set.

**11.2.3.2.0.16.7 uint32\_t dspi\_master\_ctar\_config\_t::lastSckToPcsDelayInNanoSec**

**11.2.3.2.0.16.8 uint32\_t dspi\_master\_ctar\_config\_t::betweenTransferDelayInNanoSec**

### 11.2.3.3 struct dspi\_master\_config\_t

#### Data Fields

- **dspi\_ctar\_selection\_t whichCtar**  
*Desired CTAR to use.*
- **dspi\_master\_ctar\_config\_t ctarConfig**  
*Set the ctarConfig to the desired CTAR.*
- **dspi\_which\_pcs\_t whichPcs**  
*Desired Peripheral Chip Select (pcs).*
- **dspi\_pcs\_polarity\_config\_t pcsActiveHighOrLow**  
*Desired PCS active high or low.*
- **bool enableContinuousSCK**  
*CONT\_SCKE, continuous SCK enable .*
- **bool enableRxFifoOverWrite**  
*ROOE, Receive FIFO overflow overwrite enable.*
- **bool enableModifiedTimingFormat**  
*Enables a modified transfer format to be used if it's true.*
- **dspi\_master\_sample\_point\_t samplePoint**  
*Controls when the module master samples SIN in Modified Transfer Format.*

## DSPI Driver

### 11.2.3.3.0.17 Field Documentation

**11.2.3.3.0.17.1 `dspi_ctar_selection_t dspi_master_config_t::whichCtar`**

**11.2.3.3.0.17.2 `dspi_master_ctar_config_t dspi_master_config_t::ctarConfig`**

**11.2.3.3.0.17.3 `dspi_which_pcs_t dspi_master_config_t::whichPcs`**

**11.2.3.3.0.17.4 `dspi_pcs_polarity_config_t dspi_master_config_t::pcsActiveHighOrLow`**

**11.2.3.3.0.17.5 `bool dspi_master_config_t::enableContinuousSCK`**

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

**11.2.3.3.0.17.6 `bool dspi_master_config_t::enableRx_fifo_overWrite`**

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

**11.2.3.3.0.17.7 `bool dspi_master_config_t::enableModifiedTimingFormat`**

**11.2.3.3.0.17.8 `dspi_master_sample_point_t dspi_master_config_t::samplePoint`**

It's valid only when CPHA=0.

### 11.2.3.4 `struct dspi_slave_ctar_config_t`

#### Data Fields

- `uint32_t bitsPerFrame`  
*Bits per frame, minimum 4, maximum 16.*
- `dspi_clock_polarity_t cpol`  
*Clock polarity.*
- `dspi_clock_phase_t cpha`  
*Clock phase.*

### 11.2.3.4.0.18 Field Documentation

**11.2.3.4.0.18.1 `uint32_t dspi_slave_ctar_config_t::bitsPerFrame`**

**11.2.3.4.0.18.2 `dspi_clock_polarity_t dspi_slave_ctar_config_t::cpol`**

**11.2.3.4.0.18.3 `dspi_clock_phase_t dspi_slave_ctar_config_t::cpha`**

Slave only supports MSB , does not support LSB.

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

#### Data Fields

- `dspi_ctar_selection_t whichCtar`  
*Desired CTAR to use.*
- `dspi_slave_ctar_config_t ctarConfig`  
*Set the ctarConfig to the desired CTAR.*
- `bool enableContinuousSCK`  
*CONT\_SCKE, continuous SCK enable.*
- `bool enableRxFifoOverWrite`  
*ROOE, Receive FIFO overflow overwrite enable.*
- `bool enableModifiedTimingFormat`  
*Enables a modified transfer format to be used if it's true.*
- `dspi_master_sample_point_t samplePoint`  
*Controls when the module master samples SIN in Modified Transfer Format.*

#### 11.2.3.5.0.19 Field Documentation

##### 11.2.3.5.0.19.1 `dspi_ctar_selection_t dspi_slave_config_t::whichCtar`

##### 11.2.3.5.0.19.2 `dspi_slave_ctar_config_t dspi_slave_config_t::ctarConfig`

##### 11.2.3.5.0.19.3 `bool dspi_slave_config_t::enableContinuousSCK`

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

##### 11.2.3.5.0.19.4 `bool dspi_slave_config_t::enableRxFifoOverWrite`

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

##### 11.2.3.5.0.19.5 `bool dspi_slave_config_t::enableModifiedTimingFormat`

##### 11.2.3.5.0.19.6 `dspi_master_sample_point_t dspi_slave_config_t::samplePoint`

It's valid only when CPHA=0.

### 11.2.3.6 struct dspi\_transfer\_t

#### Data Fields

- `uint8_t * txData`  
*Send buffer.*
- `uint8_t * rxData`  
*Receive buffer.*
- `volatile size_t dataSize`  
*Transfer bytes.*
- `uint32_t configFlags`  
*Transfer transfer configuration flags , set from \_dspl\_transfer\_config\_flag\_for\_master if the transfer is*

## DSPI Driver

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

### 11.2.3.6.0.20 Field Documentation

11.2.3.6.0.20.1 `uint8_t* dspi_transfer_t::txData`

11.2.3.6.0.20.2 `uint8_t* dspi_transfer_t::rxData`

11.2.3.6.0.20.3 `volatile size_t dspi_transfer_t::dataSize`

11.2.3.6.0.20.4 `uint32_t dspi_transfer_t::configFlags`

### 11.2.3.7 `struct _dspi_master_handle`

Forward declaration of the `_dspi_master_handle` typedefs.

## Data Fields

- `uint32_t bitsPerFrame`  
*Desired number of bits per frame.*
- `volatile uint32_t command`  
*Desired data command.*
- `volatile uint32_t lastCommand`  
*Desired last data command.*
- `uint8_t fifoSize`  
*FIFO dataSize.*
- `volatile bool isPcsActiveAfterTransfer`  
*Is PCS signal keep active after the last frame transfer.*
- `volatile bool isThereExtraByte`  
*Is there extra byte.*
- `uint8_t *volatile txData`  
*Send buffer.*
- `uint8_t *volatile rxData`  
*Receive buffer.*
- `volatile size_t remainingSendByteCount`  
*Number of bytes remaining to send.*
- `volatile size_t remainingReceiveByteCount`  
*Number of bytes remaining to receive.*
- `size_t totalByteCount`  
*Number of transfer bytes.*
- `volatile uint8_t state`  
*DSPI transfer state , `_dspi_transfer_state`.*
- `dspi_master_transfer_callback_t callback`  
*Completion callback.*
- `void *userData`  
*Callback user data.*

### 11.2.3.7.0.21 Field Documentation

- 11.2.3.7.0.21.1 `uint32_t dspi_master_handle_t::bitsPerFrame`
- 11.2.3.7.0.21.2 `volatile uint32_t dspi_master_handle_t::command`
- 11.2.3.7.0.21.3 `volatile uint32_t dspi_master_handle_t::lastCommand`
- 11.2.3.7.0.21.4 `uint8_t dspi_master_handle_t::fifoSize`
- 11.2.3.7.0.21.5 `volatile bool dspi_master_handle_t::isPcsActiveAfterTransfer`
- 11.2.3.7.0.21.6 `volatile bool dspi_master_handle_t::isThereExtraByte`
- 11.2.3.7.0.21.7 `uint8_t* volatile dspi_master_handle_t::txData`
- 11.2.3.7.0.21.8 `uint8_t* volatile dspi_master_handle_t::rxData`
- 11.2.3.7.0.21.9 `volatile size_t dspi_master_handle_t::remainingSendByteCount`
- 11.2.3.7.0.21.10 `volatile size_t dspi_master_handle_t::remainingReceiveByteCount`
- 11.2.3.7.0.21.11 `volatile uint8_t dspi_master_handle_t::state`
- 11.2.3.7.0.21.12 `dspi_master_transfer_callback_t dspi_master_handle_t::callback`
- 11.2.3.7.0.21.13 `void* dspi_master_handle_t::userData`

### 11.2.3.8 struct \_dspi\_slave\_handle

Forward declaration of the `_dspi_slave_handle` typedefs.

### Data Fields

- `uint32_t bitsPerFrame`  
*Desired number of bits per frame.*
- `volatile bool isThereExtraByte`  
*Is there extra byte.*
- `uint8_t *volatile txData`  
*Send buffer.*
- `uint8_t *volatile rxData`  
*Receive buffer.*
- `volatile size_t remainingSendByteCount`  
*Number of bytes remaining to send.*
- `volatile size_t remainingReceiveByteCount`  
*Number of bytes remaining to receive.*
- `size_t totalByteCount`  
*Number of transfer bytes.*
- `volatile uint8_t state`  
*DSPI transfer state.*

## DSPI Driver

- volatile uint32\_t `errorCount`  
*Error count for slave transfer.*
- `dspi_slave_transfer_callback_t` `callback`  
*Completion callback.*
- void \* `userData`  
*Callback user data.*

### 11.2.3.8.0.22 Field Documentation

11.2.3.8.0.22.1 `uint32_t dspi_slave_handle_t::bitsPerFrame`

11.2.3.8.0.22.2 `volatile bool dspi_slave_handle_t::isThereExtraByte`

11.2.3.8.0.22.3 `uint8_t* volatile dspi_slave_handle_t::txData`

11.2.3.8.0.22.4 `uint8_t* volatile dspi_slave_handle_t::rxData`

11.2.3.8.0.22.5 `volatile size_t dspi_slave_handle_t::remainingSendByteCount`

11.2.3.8.0.22.6 `volatile size_t dspi_slave_handle_t::remainingReceiveByteCount`

11.2.3.8.0.22.7 `volatile uint8_t dspi_slave_handle_t::state`

11.2.3.8.0.22.8 `volatile uint32_t dspi_slave_handle_t::errorCount`

11.2.3.8.0.22.9 `dspi_slave_transfer_callback_t dspi_slave_handle_t::callback`

11.2.3.8.0.22.10 `void* dspi_slave_handle_t::userData`

### 11.2.4 Macro Definition Documentation

11.2.4.1 `#define FSL_DSPI_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`

11.2.4.2 `#define DSPI_DUMMY_DATA (0x00U)`

Dummy data used for tx if there is not txData.

- 11.2.4.3 #define DSPI\_MASTER\_CTAR\_SHIFT (0U)
- 11.2.4.4 #define DSPI\_MASTER\_CTAR\_MASK (0x0FU)
- 11.2.4.5 #define DSPI\_MASTER\_PCS\_SHIFT (4U)
- 11.2.4.6 #define DSPI\_MASTER\_PCS\_MASK (0xF0U)
- 11.2.4.7 #define DSPI\_SLAVE\_CTAR\_SHIFT (0U)
- 11.2.4.8 #define DSPI\_SLAVE\_CTAR\_MASK (0x07U)

## 11.2.5 Typedef Documentation

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

## DSPI Driver

Parameters

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

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

Parameters

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

## 11.2.6 Enumeration Type Documentation

### 11.2.6.1 `enum _dspi_status`

Enumerator

*kStatus\_DSPI\_Busy* DSPI transfer is busy.  
*kStatus\_DSPI\_Error* DSPI driver error.  
*kStatus\_DSPI\_Idle* DSPI is idle.  
*kStatus\_DSPI\_OutOfRange* DSPI transfer out Of range.

### 11.2.6.2 `enum _dspi_flags`

Enumerator

*kDSPI\_TxCompleteFlag* Transfer Complete Flag.  
*kDSPI\_EndOfQueueFlag* End of Queue Flag.  
*kDSPI\_TxFifoUnderflowFlag* Transmit FIFO Underflow Flag.  
*kDSPI\_TxFifoFillRequestFlag* Transmit FIFO Fill Flag.  
*kDSPI\_RxFifoOverflowFlag* Receive FIFO Overflow Flag.  
*kDSPI\_RxFifoDrainRequestFlag* Receive FIFO Drain Flag.  
*kDSPI\_TxAndRxStatusFlag* The module is in Stopped/Running state.  
*kDSPI\_AllStatusFlag* All status above.

### 11.2.6.3 enum \_dspi\_interrupt\_enable

Enumerator

- kDSPI\_TxCompleteInterruptEnable* TCF interrupt enable.
- kDSPI\_EndOfQueueInterruptEnable* EOQF interrupt enable.
- kDSPI\_TxFifoUnderflowInterruptEnable* TFUF interrupt enable.
- kDSPI\_TxFifoFillRequestInterruptEnable* TFFF interrupt enable, DMA disable.
- kDSPI\_RxFifoOverflowInterruptEnable* RFOF interrupt enable.
- kDSPI\_RxFifoDrainRequestInterruptEnable* RFDF interrupt enable, DMA disable.
- kDSPI\_AllInterruptEnable* All above interrupts enable.

### 11.2.6.4 enum \_dspi\_dma\_enable

Enumerator

- kDSPI\_TxDmaEnable* TFFF flag generates DMA requests. No Tx interrupt request.
- kDSPI\_RxDmaEnable* RFDF flag generates DMA requests. No Rx interrupt request.

### 11.2.6.5 enum dspi\_master\_slave\_mode\_t

Enumerator

- kDSPI\_Master* DSPI peripheral operates in master mode.
- kDSPI\_Slave* DSPI peripheral operates in slave mode.

### 11.2.6.6 enum dspi\_master\_sample\_point\_t

This field is valid only when CPHA bit in CTAR register is 0.

Enumerator

- kDSPI\_SckToSin0Clock* 0 system clocks between SCK edge and SIN sample.
- kDSPI\_SckToSin1Clock* 1 system clock between SCK edge and SIN sample.
- kDSPI\_SckToSin2Clock* 2 system clocks between SCK edge and SIN sample.

### 11.2.6.7 enum dspi\_which\_pcs\_t

Enumerator

- kDSPI\_Pcs0* Pcs[0].
- kDSPI\_Pcs1* Pcs[1].
- kDSPI\_Pcs2* Pcs[2].

## DSPI Driver

*kDSPI\_Pcs3* Pcs[3].

*kDSPI\_Pcs4* Pcs[4].

*kDSPI\_Pcs5* Pcs[5].

### 11.2.6.8 enum dspi\_pcs\_polarity\_config\_t

Enumerator

*kDSPI\_PcsActiveHigh* Pcs Active High (idles low).

*kDSPI\_PcsActiveLow* Pcs Active Low (idles high).

### 11.2.6.9 enum \_dspi\_pcs\_polarity

Enumerator

*kDSPI\_Pcs0ActiveLow* Pcs0 Active Low (idles high).

*kDSPI\_Pcs1ActiveLow* Pcs1 Active Low (idles high).

*kDSPI\_Pcs2ActiveLow* Pcs2 Active Low (idles high).

*kDSPI\_Pcs3ActiveLow* Pcs3 Active Low (idles high).

*kDSPI\_Pcs4ActiveLow* Pcs4 Active Low (idles high).

*kDSPI\_Pcs5ActiveLow* Pcs5 Active Low (idles high).

*kDSPI\_PcsAllActiveLow* Pcs0 to Pcs5 Active Low (idles high).

### 11.2.6.10 enum dspi\_clock\_polarity\_t

Enumerator

*kDSPI\_ClockPolarityActiveHigh* CPOL=0. Active-high DSPI clock (idles low).

*kDSPI\_ClockPolarityActiveLow* CPOL=1. Active-low DSPI clock (idles high).

### 11.2.6.11 enum dspi\_clock\_phase\_t

Enumerator

*kDSPI\_ClockPhaseFirstEdge* CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

*kDSPI\_ClockPhaseSecondEdge* CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

**11.2.6.12 enum dspi\_shift\_direction\_t**

Enumerator

***kDSPI\_MsbFirst*** Data transfers start with most significant bit.***kDSPI\_LsbFirst*** Data transfers start with least significant bit.**11.2.6.13 enum dspi\_delay\_type\_t**

Enumerator

***kDSPI\_PcsToSck*** Pcs-to-SCK delay.***kDSPI\_LastSckToPcs*** Last SCK edge to Pcs delay.***kDSPI\_BetweenTransfer*** Delay between transfers.**11.2.6.14 enum dspi\_ctar\_selection\_t**

Enumerator

***kDSPI\_Ctar0*** CTAR0 selection option for master or slave mode, note that CTAR0 and CTAR0\_SLAVE are the same register address.***kDSPI\_Ctar1*** CTAR1 selection option for master mode only.***kDSPI\_Ctar2*** CTAR2 selection option for master mode only , note that some device do not support CTAR2.***kDSPI\_Ctar3*** CTAR3 selection option for master mode only , note that some device do not support CTAR3.***kDSPI\_Ctar4*** CTAR4 selection option for master mode only , note that some device do not support CTAR4.***kDSPI\_Ctar5*** CTAR5 selection option for master mode only , note that some device do not support CTAR5.***kDSPI\_Ctar6*** CTAR6 selection option for master mode only , note that some device do not support CTAR6.***kDSPI\_Ctar7*** CTAR7 selection option for master mode only , note that some device do not support CTAR7.**11.2.6.15 enum \_dspi\_transfer\_config\_flag\_for\_master**

Enumerator

***kDSPI\_MasterCtar0*** DSPI master transfer use CTAR0 setting.***kDSPI\_MasterCtar1*** DSPI master transfer use CTAR1 setting.***kDSPI\_MasterCtar2*** DSPI master transfer use CTAR2 setting.***kDSPI\_MasterCtar3*** DSPI master transfer use CTAR3 setting.

## DSPI Driver

***kDSPI\_MasterCtar4*** DSPI master transfer use CTAR4 setting.  
***kDSPI\_MasterCtar5*** DSPI master transfer use CTAR5 setting.  
***kDSPI\_MasterCtar6*** DSPI master transfer use CTAR6 setting.  
***kDSPI\_MasterCtar7*** DSPI master transfer use CTAR7 setting.  
***kDSPI\_MasterPcs0*** DSPI master transfer use PCS0 signal.  
***kDSPI\_MasterPcs1*** DSPI master transfer use PCS1 signal.  
***kDSPI\_MasterPcs2*** DSPI master transfer use PCS2 signal.  
***kDSPI\_MasterPcs3*** DSPI master transfer use PCS3 signal.  
***kDSPI\_MasterPcs4*** DSPI master transfer use PCS4 signal.  
***kDSPI\_MasterPcs5*** DSPI master transfer use PCS5 signal.  
***kDSPI\_MasterPcsContinuous*** Is PCS signal continuous.  
***kDSPI\_MasterActiveAfterTransfer*** Is PCS signal active after last frame transfer.

### 11.2.6.16 enum \_dsPIC\_transfer\_config\_flag\_for\_slave

Enumerator

***kDSPI\_SlaveCtar0*** DSPI slave transfer use CTAR0 setting. DSPI slave can only use PCS0.

### 11.2.6.17 enum \_dsPIC\_transfer\_state

Enumerator

***kDSPI\_Idle*** Nothing in the transmitter/receiver.  
***kDSPI\_Busy*** Transfer queue is not finished.  
***kDSPI\_Error*** Transfer error.

## 11.2.7 Function Documentation

### 11.2.7.1 void DSPI\_MasterInit ( SPI\_Type \* *base*, const dspi\_master\_config\_t \* *masterConfig*, uint32\_t *srcClock\_Hz* )

This function initializes the DSPI master configuration. An example use case is as follows:

```
*     dspi_master_config_t masterConfig;
*     masterConfig.whichCtar          = kDSPI_Ctar0;
*     masterConfig.ctarConfig.baudRate = 500000000;
*     masterConfig.ctarConfig.bitsPerFrame = 8;
*     masterConfig.ctarConfig.cpol      =
*         kDSPI_ClockPolarityActiveHigh;
*     masterConfig.ctarConfig.cpha      =
*         kDSPI_ClockPhaseFirstEdge;
*     masterConfig.ctarConfig.direction =
*         kDSPI_MsbFirst;
*     masterConfig.ctarConfig.pcsToSckDelayInNanoSec =
*         10000000000 /
*         masterConfig.ctarConfig.baudRate ;
```

```
*     masterConfig.ctarConfig.lastSckToPcsDelayInNanoSec      = 10000000000
*           / masterConfig.ctarConfig.baudRate ;
*     masterConfig.ctarConfig.betweenTransferDelayInNanoSec   =
*           1000000000 / masterConfig.ctarConfig.baudRate ;
*     masterConfig.whichPcs                                     = kDSPI_Pcs0
*     masterConfig.pcsActiveHighOrLow                         =
*           kDSPI_PcsActiveLow;
*     masterConfig.enableContinuousSCK                        = false;
*     masterConfig.enableRx_fifoOverWrite                   = false;
*     masterConfig.enableModifiedTimingFormat             = false;
*     masterConfig.samplePoint                            =
*           kDSPI_SckToSin0Clock;
*     DSPI_MasterInit (base, &masterConfig, srcClock_Hz);
*
```

## Parameters

|                     |                                                             |
|---------------------|-------------------------------------------------------------|
| <i>base</i>         | DSPI peripheral address.                                    |
| <i>masterConfig</i> | Pointer to structure <a href="#">dspi_master_config_t</a> . |
| <i>srcClock_Hz</i>  | Module source input clock in Hertz                          |

**11.2.7.2 void DSPI\_MasterGetDefaultConfig ( dsPIC33FJ256GP202 \* dsPIC, dspi\_master\_config\_t \* masterConfig )**

The purpose of this API is to get the configuration structure initialized for the [DSPI\\_MasterInit\(\)](#). User may use the initialized structure unchanged in [DSPI\\_MasterInit\(\)](#) or modify the structure before calling [DSPI\\_MasterInit\(\)](#). Example:

```
*     dspi_master_config_t masterConfig;
*     DSPI_MasterGetDefaultConfig(&masterConfig);
*
```

## Parameters

*masterConfig* pointer to [dspi\\_master\\_config\\_t](#) structure

11.2.7.3 void DSPI\_SlaveInit ( SPI\_Type \* *base*, const dspi\_slave\_config\_t \* *slaveConfig* )

This function initializes the DSPI slave configuration. An example use case is as follows:

```

*   dspi_slave_config_t slaveConfig;
* slaveConfig->whichCtar          = kDSPI_Ctar0;
* slaveConfig->ctarConfig.bitsPerFrame = 8;
* slaveConfig->ctarConfig.cpol      =
*               kDSPI_ClockPolarityActiveHigh;
* slaveConfig->ctarConfig.cpha      =
*               kDSPI_ClockPhaseFirstEdge;
* slaveConfig->enableContinuousSCK = false;
* slaveConfig->enableRx_fifoOverWrite = false;
* slaveConfig->enableModifiedTimingFormat = false;
* slaveConfig->samplePoint        = kDSPI_SckToSin0Clock;
*   DSPI_SlaveInit(base, &slaveConfig);
*

```

## DSPI Driver

Parameters

|                    |                                                            |
|--------------------|------------------------------------------------------------|
| <i>base</i>        | DSPI peripheral address.                                   |
| <i>slaveConfig</i> | Pointer to structure <a href="#">dspi_slave_config_t</a> . |

### 11.2.7.4 void DSPI\_SlaveGetDefaultConfig ( [dspi\\_slave\\_config\\_t](#) \* *slaveConfig* )

The purpose of this API is to get the configuration structure initialized for the [DSPI\\_SlaveInit\(\)](#). User may use the initialized structure unchanged in [DSPI\\_SlaveInit\(\)](#), or modify the structure before calling [DSPI\\_SlaveInit\(\)](#). Example:

```
* dspi\_slave\_config\_t slaveConfig;
* DSPI_SlaveGetDefaultConfig(&slaveConfig);
*
```

Parameters

|                    |                                                           |
|--------------------|-----------------------------------------------------------|
| <i>slaveConfig</i> | pointer to <a href="#">dspi_slave_config_t</a> structure. |
|--------------------|-----------------------------------------------------------|

### 11.2.7.5 void DSPI\_Deinit ( [SPI\\_Type](#) \* *base* )

Call this API to disable the DSPI clock.

Parameters

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

### 11.2.7.6 static void DSPI\_Enable ( [SPI\\_Type](#) \* *base*, [bool](#) *enable* ) [inline], [static]

Parameters

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>base</i>   | DSPI peripheral address.                             |
| <i>enable</i> | pass true to enable module, false to disable module. |

### 11.2.7.7 static [uint32\\_t](#) DSPI\_GetStatusFlags ( [SPI\\_Type](#) \* *base* ) [inline], [static]

## Parameters

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

## Returns

The DSPI status(in SR register).

### 11.2.7.8 static void DSPI\_ClearStatusFlags ( SPI\_Type \* *base*, uint32\_t *statusFlags* ) [inline], [static]

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status bit to clear. The list of status bits is defined in the dspi\_status\_and\_interrupt\_request\_t. The function uses these bit positions in its algorithm to clear the desired flag state. Example usage:

```
* DSPI_ClearStatusFlags(base, kDSPI_TxCompleteFlag |
    kDSPI_EndOfQueueFlag);
*
```

## Parameters

|                    |                                              |
|--------------------|----------------------------------------------|
| <i>base</i>        | DSPI peripheral address.                     |
| <i>statusFlags</i> | The status flag , used from type dspi_flags. |

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

### 11.2.7.9 void DSPI\_EnableInterrupts ( SPI\_Type \* *base*, uint32\_t *mask* )

This function configures the various interrupt masks of the DSPI. The parameters are base and an interrupt mask. Note, for Tx Fill and Rx FIFO drain requests, enable the interrupt request and disable the DMA request.

```
* DSPI_EnableInterrupts(base,
    kDSPI_TxCompleteInterruptEnable |
    kDSPI_EndOfQueueInterruptEnable );
*
```

## Parameters

## DSPI Driver

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | DSPI peripheral address.                                     |
| <i>mask</i> | The interrupt mask, can use the enum _dspi_interrupt_enable. |

### 11.2.7.10 static void DSPI\_DisableInterrupts ( SPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

```
* DSPI_DisableInterrupts(base,  
    kDSPI_TxCompleteInterruptEnable |  
    kDSPI_EndOfQueueInterruptEnable );  
*
```

Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | DSPI peripheral address.                                     |
| <i>mask</i> | The interrupt mask, can use the enum _dspi_interrupt_enable. |

### 11.2.7.11 static void DSPI\_EnableDMA ( SPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function configures the Rx and Tx DMA mask of the DSPI. The parameters are base and a DMA mask.

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

Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | DSPI peripheral address.                             |
| <i>mask</i> | The interrupt mask can use the enum dspi_dma_enable. |

### 11.2.7.12 static void DSPI\_DisableDMA ( SPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function configures the Rx and Tx DMA mask of the DSPI. The parameters are base and a DMA mask.

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

Parameters

|             |                                                                    |
|-------------|--------------------------------------------------------------------|
| <i>base</i> | DSPI peripheral address.                                           |
| <i>mask</i> | The interrupt mask can use the enum <code>dspi_dma_enable</code> . |

#### **11.2.7.13 static uint32\_t DSPI\_MasterGetTxRegisterAddress ( SPI\_Type \* *base* ) [inline], [static]**

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

Parameters

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

Returns

The DSPI master PUSHR data register address.

#### **11.2.7.14 static uint32\_t DSPI\_SlaveGetTxRegisterAddress ( SPI\_Type \* *base* ) [inline], [static]**

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

Parameters

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

Returns

The DSPI slave PUSHR data register address.

#### **11.2.7.15 static uint32\_t DSPI\_GetRxRegisterAddress ( SPI\_Type \* *base* ) [inline], [static]**

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

## DSPI Driver

Parameters

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

Returns

The DSPI POPR data register address.

### 11.2.7.16 static void DSPI\_SetMasterSlaveMode ( SPI\_Type \* *base*, dspi\_master\_slave\_mode\_t *mode* ) [inline], [static]

Parameters

|             |                                                                  |
|-------------|------------------------------------------------------------------|
| <i>base</i> | DSPI peripheral address.                                         |
| <i>mode</i> | Mode setting (master or slave) of type dspi_master_slave_mode_t. |

### 11.2.7.17 static bool DSPI\_IsMaster ( SPI\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

### 11.2.7.18 static void DSPI\_StartTransfer ( SPI\_Type \* *base* ) [inline], [static]

This function sets the module to begin data transfer in either master or slave mode.

Parameters

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

### 11.2.7.19 static void DSPI\_StopTransfer ( SPI\_Type \* *base* ) [inline], [static]

This function stops data transfers in either master or slave mode.

Parameters

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

#### 11.2.7.20 static void DSPI\_SetFifoEnable ( SPI\_Type \* *base*, bool *enableTxFifo*, bool *enableRxFifo* ) [inline], [static]

This function allows the caller to disable/enable the Tx and Rx FIFOs (independently). Note that to disable, the caller must pass in a logic 0 (false) for the particular FIFO configuration. To enable, the caller must pass in a logic 1 (true).

Parameters

|                     |                                                               |
|---------------------|---------------------------------------------------------------|
| <i>base</i>         | DSPI peripheral address.                                      |
| <i>enableTxFifo</i> | Disables (false) the TX FIFO, else enables (true) the TX FIFO |
| <i>enableRxFifo</i> | Disables (false) the RX FIFO, else enables (true) the RX FIFO |

#### 11.2.7.21 static void DSPI\_FlushFifo ( SPI\_Type \* *base*, bool *flushTxFifo*, bool *flushRxFifo* ) [inline], [static]

Parameters

|                    |                                                                   |
|--------------------|-------------------------------------------------------------------|
| <i>base</i>        | DSPI peripheral address.                                          |
| <i>flushTxFifo</i> | Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO |
| <i>flushRxFifo</i> | Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO |

#### 11.2.7.22 static void DSPI\_SetAllPcsPolarity ( SPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

For example, PCS0 and PCS1 set to active low and other PCS set to active high. Note that the number of PCSs is specific to the device.

```
* DSPI_SetAllPcsPolarity(base, kDSPI_Pcs0ActiveLow |
    kDSPI_Pcs1ActiveLow);
```

## DSPI Driver

Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | DSPI peripheral address.                                     |
| <i>mask</i> | The PCS polarity mask , can use the enum _dspi_pcs_polarity. |

### 11.2.7.23 `uint32_t DSPI_MasterSetBaudRate ( SPI_Type * base, dspi_ctar_selection_t whichCtar, uint32_t baudRate_Bps, uint32_t srcClock_Hz )`

This function takes in the desired baudRate\_Bps (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate, and returns the calculated baud rate in bits-per-second. It requires that the caller also provide the frequency of the module source clock (in Hertz).

Parameters

|                     |                                                                                             |
|---------------------|---------------------------------------------------------------------------------------------|
| <i>base</i>         | DSPI peripheral address.                                                                    |
| <i>whichCtar</i>    | The desired Clock and Transfer Attributes Register (CTAR) of the type dspi_ctar_selection_t |
| <i>baudRate_Bps</i> | The desired baud rate in bits per second                                                    |
| <i>srcClock_Hz</i>  | Module source input clock in Hertz                                                          |

Returns

The actual calculated baud rate

### 11.2.7.24 `void DSPI_MasterSetDelayScaler ( SPI_Type * base, dspi_ctar_selection_t whichCtar, uint32_t prescaler, uint32_t scaler, dspi_delay_type_t whichDelay )`

This function configures the PCS to SCK delay pre-scalar (PcsSCK) and scalar (CSSCK), after SCK delay pre-scalar (PASC) and scalar (ASC), and the delay after transfer pre-scalar (PDT)and scalar (DT).

These delay names are available in type dspi\_delay\_type\_t.

The user passes the delay to configure along with the prescaler and scalar value. This allows the user to directly set the prescaler/scalar values if they have pre-calculated them or if they simply wish to manually increment either value.

Parameters

|                   |                                                                                                        |
|-------------------|--------------------------------------------------------------------------------------------------------|
| <i>base</i>       | DSPI peripheral address.                                                                               |
| <i>whichCtar</i>  | The desired Clock and Transfer Attributes Register (CTAR) of type <code>dspi_ctar_selection_t</code> . |
| <i>prescaler</i>  | The prescaler delay value (can be an integer 0, 1, 2, or 3).                                           |
| <i>scaler</i>     | The scaler delay value (can be any integer between 0 to 15).                                           |
| <i>whichDelay</i> | The desired delay to configure, must be of type <code>dspi_delay_type_t</code>                         |

**11.2.7.25 `uint32_t DSPI_MasterSetDelayTimes ( SPI_Type * base, dspi_ctar_selection_t whichCtar, dspi_delay_type_t whichDelay, uint32_t srcClock_Hz, uint32_t delayTimeInNanoSec )`**

This function calculates the values for: PCS to SCK delay pre-scalar (PCSSCK) and scalar (CSSCK), or After SCK delay pre-scalar (PASC) and scalar (ASC), or Delay after transfer pre-scalar (PDT)and scalar (DT).

These delay names are available in type `dspi_delay_type_t`.

The user passes which delay they want to configure along with the desired delay value in nanoseconds. The function calculates the values needed for the prescaler and scaler and returning the actual calculated delay as an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. The higher-level peripheral driver alerts the user of an out of range delay input.

Parameters

|                            |                                                                                                        |
|----------------------------|--------------------------------------------------------------------------------------------------------|
| <i>base</i>                | DSPI peripheral address.                                                                               |
| <i>whichCtar</i>           | The desired Clock and Transfer Attributes Register (CTAR) of type <code>dspi_ctar_selection_t</code> . |
| <i>whichDelay</i>          | The desired delay to configure, must be of type <code>dspi_delay_type_t</code>                         |
| <i>srcClock_Hz</i>         | Module source input clock in Hertz                                                                     |
| <i>delayTimeIn-NanoSec</i> | The desired delay value in nanoseconds.                                                                |

## DSPI Driver

Returns

The actual calculated delay value.

### 11.2.7.26 static void DSPI\_MasterWriteData ( SPI\_Type \* *base*, dspi\_command\_data\_config\_t \* *command*, uint16\_t *data* ) [inline], [static]

In master mode, the 16-bit data is appended to the 16-bit command info. The command portion provides characteristics of the data such as the optional continuous chip select operation between transfers, the desired Clock and Transfer Attributes register to use for the associated SPI frame, the desired PCS signal to use for the data transfer, whether the current transfer is the last in the queue, and whether to clear the transfer count (normally needed when sending the first frame of a data packet). This is an example:

```
*     dspi_command_data_config_t commandConfig;
*     commandConfig.isPcsContinuous = true;
*     commandConfig.whichCtar = kDSPI_Ctar0;
*     commandConfig.whichPcs = kDSPI_Pcs0;
*     commandConfig.clearTransferCount = false;
*     commandConfig.isEndOfQueue = false;
*     DSPI_MasterWriteData(base, &commandConfig, dataWord);
```

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | DSPI peripheral address.      |
| <i>command</i> | Pointer to command structure. |
| <i>data</i>    | The data word to be sent.     |

### 11.2.7.27 void DSPI\_GetDefaultDataCommandConfig ( dspi\_command\_data\_config\_t \* *command* )

The purpose of this API is to get the configuration structure initialized for use in the DSPI\_MasterWrite\_xx(). User may use the initialized structure unchanged in DSPI\_MasterWrite\_xx() or modify the structure before calling DSPI\_MasterWrite\_xx(). Example:

```
*     dspi_command_data_config_t command;
*     DSPI_GetDefaultDataCommandConfig(&command);
*
```

Parameters

|                |                                                                   |
|----------------|-------------------------------------------------------------------|
| <i>command</i> | pointer to <a href="#">dsPIC_Command_Data_Config_t</a> structure. |
|----------------|-------------------------------------------------------------------|

#### 11.2.7.28 void DSPI\_MasterWriteDataBlocking ( SPI\_Type \* *base*, dsPIC\_Command\_Data\_Config\_t \* *command*, uint16\_t *data* )

In master mode, the 16-bit data is appended to the 16-bit command info. The command portion provides characteristics of the data such as the optional continuous chip select operation between transfers, the desired Clock and Transfer Attributes register to use for the associated SPI frame, the desired PCS signal to use for the data transfer, whether the current transfer is the last in the queue, and whether to clear the transfer count (normally needed when sending the first frame of a data packet). This is an example:

```
* dsPIC_Command_Config_t commandConfig;
* commandConfig.isPcsContinuous = true;
* commandConfig.whichCtar = kDSPI_Ctar0;
* commandConfig.whichPcs = kDSPI_Pcs1;
* commandConfig.clearTransferCount = false;
* commandConfig.isEndOfQueue = false;
* DSPI_MasterWriteDataBlocking(base, &commandConfig, dataWord);
*
```

Note that this function does not return until after the transmit is complete. Also note that the DSPI must be enabled and running to transmit data (MCR[MDIS] & [HALT] = 0). Because the SPI is a synchronous protocol, receive data is available when transmit completes.

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | DSPI peripheral address.      |
| <i>command</i> | Pointer to command structure. |
| <i>data</i>    | The data word to be sent.     |

#### 11.2.7.29 static uint32\_t DSPI\_MasterGetFormattedCommand ( dsPIC\_Command\_Data\_Config\_t \* *command* ) [inline], [static]

This function allows the caller to pass in the data command structure and returns the command word formatted according to the DSPI PUSHR register bit field placement. The user can then "OR" the returned command word with the desired data to send and use the function DSPI\_HAL\_WriteCommandDataMastermode or DSPI\_HAL\_WriteCommandDataMastermodeBlocking to write the entire 32-bit command data word to the PUSHR. This helps improve performance in cases where the command structure is constant. For example, the user calls this function before starting a transfer to generate the command word. When they are ready to transmit the data, they OR this formatted command word with the desired data to transmit. This process increases transmit performance when compared to calling send functions such as DSPI\_HAL\_WriteDataMastermode which format the command word each time a data word is to be sent.

## DSPI Driver

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>command</i> | Pointer to command structure. |
|----------------|-------------------------------|

Returns

The command word formatted to the PUSHR data register bit field.

### 11.2.7.30 void DSPI\_MasterWriteCommandDataBlocking ( SPI\_Type \* *base*, uint32\_t *data* )

In this function, the user must append the 16-bit data to the 16-bit command info then provide the total 32-bit word as the data to send. The command portion provides characteristics of the data such as the optional continuous chip select operation between transfers, the desired Clock and Transfer Attributes register to use for the associated SPI frame, the desired PCS signal to use for the data transfer, whether the current transfer is the last in the queue, and whether to clear the transfer count (normally needed when sending the first frame of a data packet). The user is responsible for appending this command with the data to send. This is an example:

```
* dataWord = <16-bit command> | <16-bit data>;
* DSPI_HAL_WriteCommandDataMastermodeBlocking(base, dataWord);
*
```

Note that this function does not return until after the transmit is complete. Also note that the DSPI must be enabled and running to transmit data (MCR[MDIS] & [HALT] = 0). Because the SPI is a synchronous protocol, the receive data is available when transmit completes.

For a blocking polling transfer, see methods below. Option 1: uint32\_t command\_to\_send = DSPI\_MasterGetFormattedCommand(&command); uint32\_t data0 = command\_to\_send | data\_need\_to\_send\_0; uint32\_t data1 = command\_to\_send | data\_need\_to\_send\_1; uint32\_t data2 = command\_to\_send | data\_need\_to\_send\_2;

```
DSPI_MasterWriteCommandDataBlocking(base,data0); DSPI_MasterWriteCommandDataBlocking(base,data1);
DSPI_MasterWriteCommandDataBlocking(base,data2);
```

Option 2: DSPI\_MasterWriteDataBlocking(base,&command,data\_need\_to\_send\_0); DSPI\_MasterWriteDataBlocking(base,&command,data\_need\_to\_send\_1); DSPI\_MasterWriteDataBlocking(base,&command,data\_need\_to\_send\_2);

Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | DSPI peripheral address.                             |
| <i>data</i> | The data word (command and data combined) to be sent |

**11.2.7.31 static void DSPI\_SlaveWriteData ( SPI\_Type \* *base*, uint32\_t *data* )  
[inline], [static]**

In slave mode, up to 16-bit words may be written.

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | DSPI peripheral address. |
| <i>data</i> | The data to send.        |

**11.2.7.32 void DSPI\_SlaveWriteDataBlocking ( SPI\_Type \* *base*, uint32\_t *data* )**

In slave mode, up to 16-bit words may be written. The function first clears the transmit complete flag, writes data into data register, and finally waits until the data is transmitted.

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | DSPI peripheral address. |
| <i>data</i> | The data to send.        |

**11.2.7.33 static uint32\_t DSPI\_ReadData ( SPI\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The data from the read data buffer.

**11.2.7.34 void DSPI\_MasterTransferCreateHandle ( SPI\_Type \* *base*, dspi\_master\_handle\_t \* *handle*, dspi\_master\_transfer\_callback\_t *callback*, void \* *userData* )**

This function initializes the DSPI handle which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, call this API once to get the initialized handle.

## DSPI Driver

Parameters

|                 |                                                            |
|-----------------|------------------------------------------------------------|
| <i>base</i>     | DSPI peripheral base address.                              |
| <i>handle</i>   | DSPI handle pointer to <code>dspi_master_handle_t</code> . |
| <i>callback</i> | dspi callback.                                             |
| <i>userData</i> | callback function parameter.                               |

### 11.2.7.35 `status_t DSPI_MasterTransferBlocking ( SPI_Type * base, dspi_transfer_t * transfer )`

This function transfers data with polling. This is a blocking function, which does not return until all transfers have been completed.

Parameters

|                 |                                                    |
|-----------------|----------------------------------------------------|
| <i>base</i>     | DSPI peripheral base address.                      |
| <i>transfer</i> | pointer to <code>dspi_transfer_t</code> structure. |

Returns

status of `status_t`.

### 11.2.7.36 `status_t DSPI_MasterTransferNonBlocking ( SPI_Type * base, dspi_master_handle_t * handle, dspi_transfer_t * transfer )`

This function transfers data using interrupts. This is a non-blocking function, which returns right away. When all data have been transferred, the callback function is called.

Parameters

|                 |                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------|
| <i>base</i>     | DSPI peripheral base address.                                                           |
| <i>handle</i>   | pointer to <code>dspi_master_handle_t</code> structure which stores the transfer state. |
| <i>transfer</i> | pointer to <code>dspi_transfer_t</code> structure.                                      |

Returns

status of `status_t`.

### 11.2.7.37 **status\_t DSPI\_MasterTransferGetCount ( SPI\_Type \* *base*, dspi\_master\_handle\_t \* *handle*, size\_t \* *count* )**

This function gets the master transfer count.

## DSPI Driver

Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                           |
| <i>handle</i> | pointer to <code>dspi_master_handle_t</code> structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.                     |

Returns

status of `status_t`.

### 11.2.7.38 void DSPI\_MasterTransferAbort ( `SPI_Type * base`, `dspi_master_handle_t * handle` )

This function aborts a transfer using an interrupt.

Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                           |
| <i>handle</i> | pointer to <code>dspi_master_handle_t</code> structure which stores the transfer state. |

### 11.2.7.39 void DSPI\_MasterTransferHandleIRQ ( `SPI_Type * base`, `dspi_master_handle_t * handle` )

This function processes the DSPI transmit and receive IRQ.

Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                           |
| <i>handle</i> | pointer to <code>dspi_master_handle_t</code> structure which stores the transfer state. |

### 11.2.7.40 void DSPI\_SlaveTransferCreateHandle ( `SPI_Type * base`, `dspi_slave_handle_t * handle`, `dspi_slave_transfer_callback_t callback`, `void * userData` )

This function initializes the DSPI handle, which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, call this API once to get the initialized handle.

Parameters

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>handle</i>   | DSPI handle pointer to <code>dspi_slave_handle_t</code> . |
| <i>base</i>     | DSPI peripheral base address.                             |
| <i>callback</i> | DSPI callback.                                            |
| <i>userData</i> | callback function parameter.                              |

#### **11.2.7.41 `status_t DSPI_SlaveTransferNonBlocking ( SPI_Type * base, dspi_slave_handle_t * handle, dspi_transfer_t * transfer )`**

This function transfers data using an interrupt. This is a non-blocking function, which returns right away. When all data have been transferred, the callback function is called.

Parameters

|                 |                                                                                        |
|-----------------|----------------------------------------------------------------------------------------|
| <i>base</i>     | DSPI peripheral base address.                                                          |
| <i>handle</i>   | pointer to <code>dspi_slave_handle_t</code> structure which stores the transfer state. |
| <i>transfer</i> | pointer to <code>dspi_transfer_t</code> structure.                                     |

Returns

status of `status_t`.

#### **11.2.7.42 `status_t DSPI_SlaveTransferGetCount ( SPI_Type * base, dspi_slave_handle_t * handle, size_t * count )`**

This function gets the slave transfer count.

Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                           |
| <i>handle</i> | pointer to <code>dspi_master_handle_t</code> structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.                     |

Returns

status of `status_t`.

### 11.2.7.43 void DSPI\_SlaveTransferAbort ( SPI\_Type \* *base*, dspi\_slave\_handle\_t \* *handle* )

This function aborts transfer using an interrupt.

Parameters

|               |                                                                                        |
|---------------|----------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                          |
| <i>handle</i> | pointer to <code>dspi_slave_handle_t</code> structure which stores the transfer state. |

#### 11.2.7.44 `void DSPI_SlaveTransferHandleIRQ ( SPI_Type * base, dspi_slave_handle_t * handle )`

This function processes the DSPI transmit and receive IRQ.

Parameters

|               |                                                                                        |
|---------------|----------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                          |
| <i>handle</i> | pointer to <code>dspi_slave_handle_t</code> structure which stores the transfer state. |

### 11.3 DSPI DMA Driver

#### 11.3.1 Overview

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

## Data Structures

- struct [dspi\\_master\\_dma\\_handle\\_t](#)  
*DSPI master DMA transfer handle structure used for transactional API. [More...](#)*
- struct [dspi\\_slave\\_dma\\_handle\\_t](#)  
*DSPI slave DMA transfer handle structure used for transactional API. [More...](#)*

## TypeDefs

- typedef void(\* [dspi\\_master\\_dma\\_transfer\\_callback\\_t](#) )(SPI\_Type \*base, dspi\_master\_dma\_handle\_t \*handle, status\_t status, void \*userData)  
*Completion callback function pointer type.*
- typedef void(\* [dspi\\_slave\\_dma\\_transfer\\_callback\\_t](#) )(SPI\_Type \*base, dspi\_slave\_dma\_handle\_t \*handle, status\_t status, void \*userData)  
*Completion callback function pointer type.*

## Functions

- void [DSPI\\_MasterTransferCreateHandleDMA](#) (SPI\_Type \*base, dspi\_master\_dma\_handle\_t \*handle, [dspi\\_master\\_dma\\_transfer\\_callback\\_t](#) callback, void \*userData, dma\_handle\_t \*dmaRxRegToRxDataHandle, dma\_handle\_t \*dmaTxDataToIntermediaryHandle, dma\_handle\_t \*dmaIntermediaryToTxRegHandle)  
*Initializes the DSPI master DMA handle.*
- status\_t [DSPI\\_MasterTransferDMA](#) (SPI\_Type \*base, dspi\_master\_dma\_handle\_t \*handle, [dspi\\_transfer\\_t](#) \*transfer)  
*DSPI master transfers data using DMA.*
- void [DSPI\\_MasterTransferAbortDMA](#) (SPI\_Type \*base, dspi\_master\_dma\_handle\_t \*handle)  
*DSPI master aborts a transfer which is using DMA.*
- status\_t [DSPI\\_MasterTransferGetCountDMA](#) (SPI\_Type \*base, dspi\_master\_dma\_handle\_t \*handle, size\_t \*count)  
*Gets the master DMA transfer remaining bytes.*
- void [DSPI\\_SlaveTransferCreateHandleDMA](#) (SPI\_Type \*base, dspi\_slave\_dma\_handle\_t \*handle, [dspi\\_slave\\_dma\\_transfer\\_callback\\_t](#) callback, void \*userData, dma\_handle\_t \*dmaRxRegToRxDataHandle, dma\_handle\_t \*dmaTxDataToTxRegHandle)  
*Initializes the DSPI slave DMA handle.*
- status\_t [DSPI\\_SlaveTransferDMA](#) (SPI\_Type \*base, dspi\_slave\_dma\_handle\_t \*handle, [dspi\\_transfer\\_t](#) \*transfer)

*DSPI slave transfers data using DMA.*

- void [DSPI\\_SlaveTransferAbortDMA](#) (SPI\_Type \*base, dspi\_slave\_dma\_handle\_t \*handle)  
*DSPI slave aborts a transfer which is using DMA.*
- status\_t [DSPI\\_SlaveTransferGetCountDMA](#) (SPI\_Type \*base, dspi\_slave\_dma\_handle\_t \*handle, size\_t \*count)  
*Gets the slave DMA transfer remaining bytes.*

## 11.3.2 Data Structure Documentation

### 11.3.2.1 struct \_dspi\_master\_dma\_handle

Forward declaration of the DSPI DMA master handle typedefs.

#### Data Fields

- uint32\_t **bitsPerFrame**  
*Desired number of bits per frame.*
- volatile uint32\_t **command**  
*Desired data command.*
- volatile uint32\_t **lastCommand**  
*Desired last data command.*
- uint8\_t **fifoSize**  
*FIFO dataSize.*
- volatile bool **isPcsActiveAfterTransfer**  
*Is PCS signal keep active after the last frame transfer.*
- volatile bool **isThereExtraByte**  
*Is there extra byte.*
- uint8\_t \*volatile **txData**  
*Send buffer.*
- uint8\_t \*volatile **rxData**  
*Receive buffer.*
- volatile size\_t **remainingSendByteCount**  
*Number of bytes remaining to send.*
- volatile size\_t **remainingReceiveByteCount**  
*Number of bytes remaining to receive.*
- size\_t **totalByteCount**  
*Number of transfer bytes.*
- uint32\_t **rxBuffIfNull**  
*Used if there is not rxData for DMA purpose.*
- uint32\_t **txBuffIfNull**  
*Used if there is not txData for DMA purpose.*
- volatile uint8\_t **state**  
*DSPI transfer state , \_dspi\_transfer\_state.*
- [dspi\\_master\\_dma\\_transfer\\_callback\\_t](#) **callback**  
*Completion callback.*
- void \* **userData**  
*Callback user data.*
- dma\_handle\_t \* **dmaRxRegToRxDataHandle**

## DSPI DMA Driver

- *dma\_handle\_t handle point used for RxReg to RxData buff*
- `dma_handle_t * dmaTxDataToIntermediaryHandle`  
*dma\_handle\_t handle point used for TxData to Intermediary*
- `dma_handle_t * dmaIntermediaryToTxRegHandle`  
*dma\_handle\_t handle point used for Intermediary to TxReg*

### 11.3.2.1.0.23 Field Documentation

11.3.2.1.0.23.1 `uint32_t dspl_master_dma_handle_t::bitsPerFrame`

11.3.2.1.0.23.2 `volatile uint32_t dspl_master_dma_handle_t::command`

11.3.2.1.0.23.3 `volatile uint32_t dspl_master_dma_handle_t::lastCommand`

11.3.2.1.0.23.4 `uint8_t dspl_master_dma_handle_t::fifoSize`

11.3.2.1.0.23.5 `volatile bool dspl_master_dma_handle_t::isPcsActiveAfterTransfer`

11.3.2.1.0.23.6 `volatile bool dspl_master_dma_handle_t::isThereExtraByte`

11.3.2.1.0.23.7 `uint8_t* volatile dspl_master_dma_handle_t::txData`

11.3.2.1.0.23.8 `uint8_t* volatile dspl_master_dma_handle_t::rxData`

11.3.2.1.0.23.9 `volatile size_t dspl_master_dma_handle_t::remainingSendByteCount`

11.3.2.1.0.23.10 `volatile size_t dspl_master_dma_handle_t::remainingReceiveByteCount`

11.3.2.1.0.23.11 `uint32_t dspl_master_dma_handle_t::rxBuffIfNull`

11.3.2.1.0.23.12 `uint32_t dspl_master_dma_handle_t::txBuffIfNull`

11.3.2.1.0.23.13 `volatile uint8_t dspl_master_dma_handle_t::state`

11.3.2.1.0.23.14 `dspl_master_dma_transfer_callback_t dspl_master_dma_handle_t::callback`

11.3.2.1.0.23.15 `void* dspl_master_dma_handle_t::userData`

### 11.3.2.2 `struct _dspl_slave_dma_handle`

Forward declaration of the DSPI DMA slave handle typedefs.

## Data Fields

- `uint32_t bitsPerFrame`  
*Desired number of bits per frame.*
- `volatile bool isThereExtraByte`  
*Is there extra byte.*
- `uint8_t *volatile txData`  
*Send buffer.*

- `uint8_t *volatile rxData`  
*Receive buffer.*
- `volatile size_t remainingSendByteCount`  
*Number of bytes remaining to send.*
- `volatile size_t remainingReceiveByteCount`  
*Number of bytes remaining to receive.*
- `size_t totalByteCount`  
*Number of transfer bytes.*
- `uint32_t rxBuffIfNull`  
*Used if there is not rxData for DMA purpose.*
- `uint32_t txBuffIfNull`  
*Used if there is not txData for DMA purpose.*
- `uint32_t txLastData`  
*Used if there is an extra byte when 16 bits per frame for DMA purpose.*
- `volatile uint8_t state`  
*DSPI transfer state.*
- `uint32_t errorCount`  
*Error count for slave transfer.*
- `dspi_slave_dma_transfer_callback_t callback`  
*Completion callback.*
- `void *userData`  
*Callback user data.*
- `dma_handle_t * dmaRxRegToRxDataHandle`  
*dma\_handle\_t handle point used for RxReg to RxData buff*
- `dma_handle_t * dmaTxDataToTxRegHandle`  
*dma\_handle\_t handle point used for TxData to TxReg*

## DSPI DMA Driver

### 11.3.2.2.0.24 Field Documentation

11.3.2.2.0.24.1 `uint32_t dspi_slave_dma_handle_t::bitsPerFrame`

11.3.2.2.0.24.2 `volatile bool dspi_slave_dma_handle_t::isThereExtraByte`

11.3.2.2.0.24.3 `uint8_t* volatile dspi_slave_dma_handle_t::txData`

11.3.2.2.0.24.4 `uint8_t* volatile dspi_slave_dma_handle_t::rxData`

11.3.2.2.0.24.5 `volatile size_t dspi_slave_dma_handle_t::remainingSendByteCount`

11.3.2.2.0.24.6 `volatile size_t dspi_slave_dma_handle_t::remainingReceiveByteCount`

11.3.2.2.0.24.7 `uint32_t dspi_slave_dma_handle_t::rxBuffIfNull`

11.3.2.2.0.24.8 `uint32_t dspi_slave_dma_handle_t::txBuffIfNull`

11.3.2.2.0.24.9 `uint32_t dspi_slave_dma_handle_t::txLastData`

11.3.2.2.0.24.10 `volatile uint8_t dspi_slave_dma_handle_t::state`

11.3.2.2.0.24.11 `uint32_t dspi_slave_dma_handle_t::errorCount`

11.3.2.2.0.24.12 `dspi_slave_dma_transfer_callback_t dspi_slave_dma_handle_t::callback`

11.3.2.2.0.24.13 `void* dspi_slave_dma_handle_t::userData`

### 11.3.3 Typedef Documentation

11.3.3.1 `typedef void(* dspi_master_dma_transfer_callback_t)(SPI_Type *base, dspi_master_dma_handle_t *handle, status_t status, void *userData)`

Parameters

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

### 11.3.3.2 **typedef void(\* dspi\_slave\_dma\_transfer\_callback\_t)(SPI\_Type \*base, dspi\_slave\_dma\_handle\_t \*handle, status\_t status, void \*userData)**

Parameters

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

## 11.3.4 Function Documentation

### 11.3.4.1 **void DSPI\_MasterTransferCreateHandleDMA ( SPI\_Type \* *base*, dspi\_master\_dma\_handle\_t \* *handle*, dspi\_master\_dma\_transfer\_callback\_t *callback*, void \* *userData*, dma\_handle\_t \* *dmaRxRegToRxDataHandle*, dma\_handle\_t \* *dmaTxDataToIntermediaryHandle*, dma\_handle\_t \* *dmaIntermediaryToTxRegHandle* )**

This function initializes the DSPI DMA handle which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, call this API once to get the initialized handle.

Note that DSPI DMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx is the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for *dmaRxRegToRxDataHandle* and Tx DMAMUX source for *dmaIntermediaryToTxRegHandle*. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for *dmaRxRegToRxDataHandle*.

Parameters

## DSPI DMA Driver

|                                        |                                                                                   |
|----------------------------------------|-----------------------------------------------------------------------------------|
| <i>base</i>                            | DSPI peripheral base address.                                                     |
| <i>handle</i>                          | DSPI handle pointer to <code>dspi_master_dma_handle_t</code> .                    |
| <i>callback</i>                        | DSPI callback.                                                                    |
| <i>userData</i>                        | callback function parameter.                                                      |
| <i>dmaRxRegTo-RxDataHandle</i>         | <code>dmaRxRegToRxDataHandle</code> pointer to <code>dma_handle_t</code> .        |
| <i>dmaTxDataTo-Intermediary-Handle</i> | <code>dmaTxDataToIntermediaryHandle</code> pointer to <code>dma_handle_t</code> . |
| <i>dma-Intermediary-ToTxReg-Handle</i> | <code>dmaIntermediaryToTxRegHandle</code> pointer to <code>dma_handle_t</code> .  |

### 11.3.4.2 `status_t DSPI_MasterTransferDMA ( SPI_Type * base, dspi_master_dma_handle_t * handle, dspi_transfer_t * transfer )`

This function transfers data using DMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note that master DMA transfer cannot support the transfer\_size of 1 when the bitsPerFrame is greater than 8.

Parameters

|                 |                                                                                             |
|-----------------|---------------------------------------------------------------------------------------------|
| <i>base</i>     | DSPI peripheral base address.                                                               |
| <i>handle</i>   | pointer to <code>dspi_master_dma_handle_t</code> structure which stores the transfer state. |
| <i>transfer</i> | pointer to <code>dspi_transfer_t</code> structure.                                          |

Returns

status of `status_t`.

### 11.3.4.3 `void DSPI_MasterTransferAbortDMA ( SPI_Type * base, dspi_master_dma_handle_t * handle )`

This function aborts a transfer which is using DMA.

Parameters

|               |                                                                                             |
|---------------|---------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                               |
| <i>handle</i> | pointer to <code>dspi_master_dma_handle_t</code> structure which stores the transfer state. |

#### 11.3.4.4 `status_t DSPI_MasterTransferGetCountDMA ( SPI_Type * base, dspi_master_dma_handle_t * handle, size_t * count )`

This function gets the master DMA transfer remaining bytes.

Parameters

|               |                                                                                             |
|---------------|---------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                               |
| <i>handle</i> | pointer to <code>dspi_master_dma_handle_t</code> structure which stores the transfer state. |
| <i>count</i>  | number point of bytes transferred so far by the non-blocking transaction.                   |

Returns

status of `status_t`.

#### 11.3.4.5 `void DSPI_SlaveTransferCreateHandleDMA ( SPI_Type * base, dspi_slave_dma_handle_t * handle, dspi_slave_dma_transfer_callback_t callback, void * userData, dma_handle_t * dmaRxRegToRxDataHandle, dma_handle_t * dmaTxDataToTxRegHandle )`

This function initializes the DSPI DMA handle which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, call this API one time to get the initialized handle.

Note that DSPI DMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx is the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for `dmaRxRegToRxDataHandle` and Tx DMAMUX source for `dmaTxDataToTxRegHandle`. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for `dmaRxRegToRxDataHandle`.

Parameters

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

## DSPI DMA Driver

|                                |                                                 |
|--------------------------------|-------------------------------------------------|
| <i>handle</i>                  | DSPI handle pointer to dspi_slave_dma_handle_t. |
| <i>callback</i>                | DSPI callback.                                  |
| <i>userData</i>                | callback function parameter.                    |
| <i>dmaRxRegTo-RxDataHandle</i> | dmaRxRegToRxDataHandle pointer to dma_handle_t. |
| <i>dmaTxDataTo-TxRegHandle</i> | dmaTxDataToTxRegHandle pointer to dma_handle_t. |

### 11.3.4.6 status\_t DSPI\_SlaveTransferDMA ( SPI\_Type \* *base*, dspi\_slave\_dma\_handle\_t \* *handle*, dspi\_transfer\_t \* *transfer* )

This function transfers data using DMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note that the slave DMA transfer cannot support the transfer\_size of 1 when the bitsPerFrame is greater than 8.

Parameters

|                 |                                                                               |
|-----------------|-------------------------------------------------------------------------------|
| <i>base</i>     | DSPI peripheral base address.                                                 |
| <i>handle</i>   | pointer to dspi_slave_dma_handle_t structure which stores the transfer state. |
| <i>transfer</i> | pointer to <a href="#">dspi_transfer_t</a> structure.                         |

Returns

status of status\_t.

### 11.3.4.7 void DSPI\_SlaveTransferAbortDMA ( SPI\_Type \* *base*, dspi\_slave\_dma\_handle\_t \* *handle* )

This function aborts a transfer which is using DMA.

Parameters

|               |                                                                               |
|---------------|-------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                 |
| <i>handle</i> | pointer to dspi_slave_dma_handle_t structure which stores the transfer state. |

### 11.3.4.8 status\_t DSPI\_SlaveTransferGetCountDMA ( SPI\_Type \* *base*, dspi\_slave\_dma\_handle\_t \* *handle*, size\_t \* *count* )

This function gets the slave DMA transfer remaining bytes.

## Parameters

|               |                                                                                            |
|---------------|--------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                              |
| <i>handle</i> | pointer to <code>dspi_slave_dma_handle_t</code> structure which stores the transfer state. |
| <i>count</i>  | number point of bytes transferred so far by the non-blocking transaction.                  |

## Returns

status of `status_t`.

## DSPI eDMA Driver

### 11.4 DSPI eDMA Driver

#### 11.4.1 Overview

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

## Data Structures

- struct [dspi\\_master\\_edma\\_handle\\_t](#)  
*DSPI master eDMA transfer handle structure used for transactional API. [More...](#)*
- struct [dspi\\_slave\\_edma\\_handle\\_t](#)  
*DSPI slave eDMA transfer handle structure used for transactional API. [More...](#)*

## Typedefs

- typedef void(\* [dspi\\_master\\_edma\\_transfer\\_callback\\_t](#)) (SPI\_Type \*base, dspi\_master\_edma\_handle\_t \*handle, status\_t status, void \*userData)  
*Completion callback function pointer type.*
- typedef void(\* [dspi\\_slave\\_edma\\_transfer\\_callback\\_t](#)) (SPI\_Type \*base, dspi\_slave\_edma\_handle\_t \*handle, status\_t status, void \*userData)  
*Completion callback function pointer type.*

## Functions

- void [DSPI\\_MasterTransferCreateHandleEDMA](#) (SPI\_Type \*base, dspi\_master\_edma\_handle\_t \*handle, [dspi\\_master\\_edma\\_transfer\\_callback\\_t](#) callback, void \*userData, [edma\\_handle\\_t](#) \*edmaRxRegToRxDataHandle, [edma\\_handle\\_t](#) \*edmaTxDataToIntermediaryHandle, [edma\\_handle\\_t](#) \*edmaIntermediaryToTxRegHandle)  
*Initializes the DSPI master eDMA handle.*
- status\_t [DSPI\\_MasterTransferEDMA](#) (SPI\_Type \*base, dspi\_master\_edma\_handle\_t \*handle, [dspi\\_transfer\\_t](#) \*transfer)  
*DSPI master transfer data using eDMA.*
- void [DSPI\\_MasterTransferAbortEDMA](#) (SPI\_Type \*base, dspi\_master\_edma\_handle\_t \*handle)  
*DSPI master aborts a transfer which using eDMA.*
- status\_t [DSPI\\_MasterTransferGetCountEDMA](#) (SPI\_Type \*base, dspi\_master\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets the master eDMA transfer count.*
- void [DSPI\\_SlaveTransferCreateHandleEDMA](#) (SPI\_Type \*base, dspi\_slave\_edma\_handle\_t \*handle, [dspi\\_slave\\_edma\\_transfer\\_callback\\_t](#) callback, void \*userData, [edma\\_handle\\_t](#) \*edmaRxRegToRxDataHandle, [edma\\_handle\\_t](#) \*edmaTxDataToTxRegHandle)  
*Initializes the DSPI slave eDMA handle.*
- status\_t [DSPI\\_SlaveTransferEDMA](#) (SPI\_Type \*base, dspi\_slave\_edma\_handle\_t \*handle, [dspi\\_transfer\\_t](#) \*transfer)

*DSPI slave transfer data using eDMA.*

- void **DSPI\_SlaveTransferAbortEDMA** (SPI\_Type \*base, dspi\_slave\_edma\_handle\_t \*handle)  
*DSPI slave aborts a transfer which using eDMA.*
- status\_t **DSPI\_SlaveTransferGetCountEDMA** (SPI\_Type \*base, dspi\_slave\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets the slave eDMA transfer count.*

## 11.4.2 Data Structure Documentation

### 11.4.2.1 struct \_dspi\_master\_edma\_handle

Forward declaration of the DSPI eDMA master handle typedefs.

#### Data Fields

- uint32\_t **bitsPerFrame**  
*Desired number of bits per frame.*
- volatile uint32\_t **command**  
*Desired data command.*
- volatile uint32\_t **lastCommand**  
*Desired last data command.*
- uint8\_t **fifoSize**  
*FIFO dataSize.*
- volatile bool **isPcsActiveAfterTransfer**  
*Is PCS signal keep active after the last frame transfer.*
- volatile bool **isThereExtraByte**  
*Is there extra byte.*
- uint8\_t \*volatile **txData**  
*Send buffer.*
- uint8\_t \*volatile **rxData**  
*Receive buffer.*
- volatile size\_t **remainingSendByteCount**  
*Number of bytes remaining to send.*
- volatile size\_t **remainingReceiveByteCount**  
*Number of bytes remaining to receive.*
- size\_t **totalByteCount**  
*Number of transfer bytes.*
- uint32\_t **rxBuffIfNull**  
*Used if there is not rxData for DMA purpose.*
- uint32\_t **txBuffIfNull**  
*Used if there is not txData for DMA purpose.*
- volatile uint8\_t **state**  
*DSPI transfer state , \_dspi\_transfer\_state.*
- **dspi\_master\_edma\_transfer\_callback\_t callback**  
*Completion callback.*
- void \* **userData**  
*Callback user data.*
- **edma\_handle\_t \* edmaRxRegToRxDataHandle**

## DSPI eDMA Driver

- *edma\_handle\_t* handle point used for RxReg to RxData buff
- *edma\_handle\_t* \* **edmaTxDataToIntermediaryHandle**  
*edma\_handle\_t* handle point used for TxData to Intermediary
- *edma\_handle\_t* \* **edmaIntermediaryToTxRegHandle**  
*edma\_handle\_t* handle point used for Intermediary to TxReg
- **edma\_tcd\_t** **dspiSoftwareTCD** [2]  
*SoftwareTCD*, internal used.

### 11.4.2.1.0.25 Field Documentation

**11.4.2.1.0.25.1** `uint32_t dspi_master_edma_handle_t::bitsPerFrame`

**11.4.2.1.0.25.2** `volatile uint32_t dspi_master_edma_handle_t::command`

**11.4.2.1.0.25.3** `volatile uint32_t dspi_master_edma_handle_t::lastCommand`

**11.4.2.1.0.25.4** `uint8_t dspi_master_edma_handle_t::fifoSize`

**11.4.2.1.0.25.5** `volatile bool dspi_master_edma_handle_t::isPcsActiveAfterTransfer`

**11.4.2.1.0.25.6** `volatile bool dspi_master_edma_handle_t::isThereExtraByte`

**11.4.2.1.0.25.7** `uint8_t* volatile dspi_master_edma_handle_t::txData`

**11.4.2.1.0.25.8** `uint8_t* volatile dspi_master_edma_handle_t::rxData`

**11.4.2.1.0.25.9** `volatile size_t dspi_master_edma_handle_t::remainingSendByteCount`

**11.4.2.1.0.25.10** `volatile size_t dspi_master_edma_handle_t::remainingReceiveByteCount`

**11.4.2.1.0.25.11** `uint32_t dspi_master_edma_handle_t::rxBuffIfNull`

**11.4.2.1.0.25.12** `uint32_t dspi_master_edma_handle_t::txBuffIfNull`

**11.4.2.1.0.25.13** `volatile uint8_t dspi_master_edma_handle_t::state`

**11.4.2.1.0.25.14** `dspi_master_edma_transfer_callback_t dspi_master_edma_handle_t::callback`

**11.4.2.1.0.25.15** `void* dspi_master_edma_handle_t::userData`

### 11.4.2.2 `struct _dspi_slave_edma_handle`

Forward declaration of the DSPI eDMA slave handle typedefs.

## Data Fields

- `uint32_t bitsPerFrame`  
*Desired number of bits per frame.*
- `volatile bool isThereExtraByte`  
*Is there extra byte.*

- `uint8_t *volatile txData`  
*Send buffer.*
- `uint8_t *volatile rxData`  
*Receive buffer.*
- `volatile size_t remainingSendByteCount`  
*Number of bytes remaining to send.*
- `volatile size_t remainingReceiveByteCount`  
*Number of bytes remaining to receive.*
- `size_t totalByteCount`  
*Number of transfer bytes.*
- `uint32_t rxBuffIfNull`  
*Used if there is not rxData for DMA purpose.*
- `uint32_t txBuffIfNull`  
*Used if there is not txData for DMA purpose.*
- `uint32_t txLastData`  
*Used if there is an extra byte when 16bits per frame for DMA purpose.*
- `volatile uint8_t state`  
*DSPI transfer state.*
- `uint32_t errorCount`  
*Error count for slave transfer.*
- `dspi_slave_edma_transfer_callback_t callback`  
*Completion callback.*
- `void *userData`  
*Callback user data.*
- `edma_handle_t * edmaRxRegToRxDataHandle`  
*edma\_handle\_t handle point used for RxReg to RxData buff*
- `edma_handle_t * edmaTxDataToTxRegHandle`  
*edma\_handle\_t handle point used for TxData to TxReg*
- `edma_tcd_t dsplSoftwareTCD [2]`  
*SoftwareTCD , internal used.*

## DSPI eDMA Driver

### 11.4.2.2.0.26 Field Documentation

11.4.2.2.0.26.1 `uint32_t dspi_slave_edma_handle_t::bitsPerFrame`

11.4.2.2.0.26.2 `volatile bool dspi_slave_edma_handle_t::isThereExtraByte`

11.4.2.2.0.26.3 `uint8_t* volatile dspi_slave_edma_handle_t::txData`

11.4.2.2.0.26.4 `uint8_t* volatile dspi_slave_edma_handle_t::rxData`

11.4.2.2.0.26.5 `volatile size_t dspi_slave_edma_handle_t::remainingSendByteCount`

11.4.2.2.0.26.6 `volatile size_t dspi_slave_edma_handle_t::remainingReceiveByteCount`

11.4.2.2.0.26.7 `uint32_t dspi_slave_edma_handle_t::rxBuffIfNull`

11.4.2.2.0.26.8 `uint32_t dspi_slave_edma_handle_t::txBuffIfNull`

11.4.2.2.0.26.9 `uint32_t dspi_slave_edma_handle_t::txLastData`

11.4.2.2.0.26.10 `volatile uint8_t dspi_slave_edma_handle_t::state`

11.4.2.2.0.26.11 `uint32_t dspi_slave_edma_handle_t::errorCount`

11.4.2.2.0.26.12 `dspi_slave_edma_transfer_callback_t dspi_slave_edma_handle_t::callback`

11.4.2.2.0.26.13 `void* dspi_slave_edma_handle_t::userData`

### 11.4.3 Typedef Documentation

11.4.3.1 `typedef void(* dspi_master_edma_transfer_callback_t)(SPI_Type *base, dspi_master_edma_handle_t *handle, status_t status, void *userData)`

Parameters

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

#### 11.4.3.2 **typedef void(\* dspi\_slave\_edma\_transfer\_callback\_t)(SPI\_Type \*base, dspi\_slave\_edma\_handle\_t \*handle, status\_t status, void \*userData)**

Parameters

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

### 11.4.4 Function Documentation

#### 11.4.4.1 **void DSPI\_MasterTransferCreateHandleEDMA ( SPI\_Type \* base, dspi\_master\_edma\_handle\_t \* handle, dspi\_master\_edma\_transfer\_callback\_t callback, void \* userData, edma\_handle\_t \* edmaRxRegToRxDataHandle, edma\_handle\_t \* edmaTxDataToIntermediaryHandle, edma\_handle\_t \* edmaIntermediaryToTxRegHandle )**

This function initializes the DSPI eDMA handle which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, user need only call this API once to get the initialized handle.

Note that DSPI eDMA has separated (RX and TX as two sources) or shared (RX and TX are the same source) DMA request source. (1)For the separated DMA request source, enable and set the RX DMAMUX source for edmaRxRegToRxDataHandle and TX DMAMUX source for edmaIntermediaryToTxRegHandle. (2)For the shared DMA request source, enable and set the RX/RX DMAMUX source for the edmaRxRegToRxDataHandle.

Parameters

## DSPI eDMA Driver

|                                          |                                                                                     |
|------------------------------------------|-------------------------------------------------------------------------------------|
| <i>base</i>                              | DSPI peripheral base address.                                                       |
| <i>handle</i>                            | DSPI handle pointer to <code>dspi_master_edma_handle_t</code> .                     |
| <i>callback</i>                          | DSPI callback.                                                                      |
| <i>userData</i>                          | callback function parameter.                                                        |
| <i>edmaRxRegTo-RxDataHandle</i>          | <code>edmaRxRegToRxDataHandle</code> pointer to <code>edma_handle_t</code> .        |
| <i>edmaTxData-To-Intermediary-Handle</i> | <code>edmaTxDataToIntermediaryHandle</code> pointer to <code>edma_handle_t</code> . |
| <i>edma-Intermediary-ToTxReg-Handle</i>  | <code>edmaIntermediaryToTxRegHandle</code> pointer to <code>edma_handle_t</code> .  |

**11.4.4.2 `status_t DSPI_MasterTransferEDMA ( SPI_Type * base, dspi_master_edma_handle_t * handle, dspi_transfer_t * transfer )`**

This function transfer data using eDMA. This is non-blocking function, which returns right away. When all data have been transfer, the callback function is called.

Parameters

|                 |                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------|
| <i>base</i>     | DSPI peripheral base address.                                                                |
| <i>handle</i>   | pointer to <code>dspi_master_edma_handle_t</code> structure which stores the transfer state. |
| <i>transfer</i> | pointer to <code>dspi_transfer_t</code> structure.                                           |

Returns

status of `status_t`.

**11.4.4.3 `void DSPI_MasterTransferAbortEDMA ( SPI_Type * base, dspi_master_edma_handle_t * handle )`**

This function aborts a transfer which using eDMA.

Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                                |
| <i>handle</i> | pointer to <code>dspi_master_edma_handle_t</code> structure which stores the transfer state. |

#### 11.4.4.4 `status_t DSPI_MasterTransferGetCountEDMA ( SPI_Type * base, dspi_master_edma_handle_t * handle, size_t * count )`

This function get the master eDMA transfer count.

Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                                |
| <i>handle</i> | pointer to <code>dspi_master_edma_handle_t</code> structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.                          |

Returns

status of `status_t`.

#### 11.4.4.5 `void DSPI_SlaveTransferCreateHandleEDMA ( SPI_Type * base, dspi_slave_edma_handle_t * handle, dspi_slave_edma_transfer_callback_t callback, void * userData, edma_handle_t * edmaRxRegToRxDataHandle, edma_handle_t * edmaTxDataToTxRegHandle )`

This function initializes the DSPI eDMA handle which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, call this API once to get the initialized handle.

Note that DSPI eDMA has separated (RN and TX in 2 sources) or shared (RX and TX are the same source) DMA request source. (1)For the separated DMA request source, enable and set the RX DMAMUX source for `edmaRxRegToRxDataHandle` and TX DMAMUX source for `edmaTxDataToTxRegHandle`. (2)For the shared DMA request source, enable and set the RX/RX DMAMUX source for the `edmaRxRegToRxDataHandle`.

Parameters

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

## DSPI eDMA Driver

|                                  |                                                                 |
|----------------------------------|-----------------------------------------------------------------|
| <i>handle</i>                    | DSPI handle pointer to <code>dspi_slave_edma_handle_t</code> .  |
| <i>callback</i>                  | DSPI callback.                                                  |
| <i>userData</i>                  | callback function parameter.                                    |
| <i>edmaRxRegTo-RxDataHandle</i>  | edmaRxRegToRxDataHandle pointer to <code>edma_handle_t</code> . |
| <i>edmaTxData-ToTxReg-Handle</i> | edmaTxDataToTxRegHandle pointer to <code>edma_handle_t</code> . |

### 11.4.4.6 `status_t DSPI_SlaveTransferEDMA ( SPI_Type * base, dspi_slave_edma_handle_t * handle, dspi_transfer_t * transfer )`

This function transfer data using eDMA. This is non-blocking function, which returns right away. When all data have been transfer, the callback function is called. Note that slave EDMA transfer cannot support the situation that transfer\_size is 1 when the bitsPerFrame is greater than 8 .

Parameters

|                 |                                                                                             |
|-----------------|---------------------------------------------------------------------------------------------|
| <i>base</i>     | DSPI peripheral base address.                                                               |
| <i>handle</i>   | pointer to <code>dspi_slave_edma_handle_t</code> structure which stores the transfer state. |
| <i>transfer</i> | pointer to <code>dspi_transfer_t</code> structure.                                          |

Returns

status of `status_t`.

### 11.4.4.7 `void DSPI_SlaveTransferAbortEDMA ( SPI_Type * base, dspi_slave_edma_handle_t * handle )`

This function aborts a transfer which using eDMA.

Parameters

|               |                                                                                             |
|---------------|---------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                               |
| <i>handle</i> | pointer to <code>dspi_slave_edma_handle_t</code> structure which stores the transfer state. |

### 11.4.4.8 `status_t DSPI_SlaveTransferGetCountEDMA ( SPI_Type * base, dspi_slave_edma_handle_t * handle, size_t * count )`

This function gets the slave eDMA transfer count.

## Parameters

|               |                                                                                             |
|---------------|---------------------------------------------------------------------------------------------|
| <i>base</i>   | DSPI peripheral base address.                                                               |
| <i>handle</i> | pointer to <code>dspi_slave_edma_handle_t</code> structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.                         |

## Returns

status of `status_t`.

### 11.5 DSPI FreeRTOS Driver

#### 11.5.1 Overview

#### Data Structures

- struct [dsPIC33F\\_DSPI\\_RTOS.h](#)  
*DSPI FreeRTOS handle.* [More...](#)

#### DSPI RTOS Operation

- status\_t [DSPI\\_RTOS\\_Init](#) (dsPIC33F\_DSPI\_RTOS.h \*handle, SPI\_Type \*base, const [dsPIC33F\\_DSPI\\_MasterConfig.h](#) \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes DSPI.*
- status\_t [DSPI\\_RTOS\\_Deinit](#) (dsPIC33F\_DSPI\_RTOS.h \*handle)  
*Deinitializes the DSPI.*
- status\_t [DSPI\\_RTOS\\_Transfer](#) (dsPIC33F\_DSPI\_RTOS.h \*handle, [dsPIC33F\\_DSPI\\_Transfer.h](#) \*transfer)  
*Performs SPI transfer.*

#### 11.5.2 Data Structure Documentation

##### 11.5.2.1 struct dsPIC33F\_DSPI\_RTOS.h

DSPI μC/OS-III handle.

DSPI μC/OS-II handle.

#### Data Fields

- SPI\_Type \* [base](#)  
*DSPI base address.*
- dsPIC33F\_DSPI\_MasterHandle\_t [drv\\_handle](#)  
*Handle of the underlying driver, treated as opaque by the RTOS layer.*
- SemaphoreHandle\_t [mutex](#)  
*Mutex to lock the handle during a transfer.*
- SemaphoreHandle\_t [event](#)  
*Semaphore to notify and unblock task when transfer ends.*
- OS\_EVENT \* [mutex](#)  
*Mutex to lock the handle during a transfer.*
- OS\_FLAG\_GRP \* [event](#)  
*Semaphore to notify and unblock task when transfer ends.*
- OS\_SEM [mutex](#)  
*Mutex to lock the handle during a transfer.*
- OS\_FLAG\_GRP [event](#)  
*Semaphore to notify and unblock task when transfer ends.*

### 11.5.3 Function Documentation

**11.5.3.1 status\_t DSPI\_RTOS\_Init ( *dspi\_rtos\_handle\_t \* handle*, *SPI\_Type \* base*, *const dspi\_master\_config\_t \* masterConfig*, *uint32\_t srcClock\_Hz* )**

This function initializes the DSPI module and the related RTOS context.

## DSPI FreeRTOS Driver

Parameters

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

Returns

status of the operation.

### 11.5.3.2 status\_t DSPI\_RTOS\_Deinit ( *dspi\_rtos\_handle\_t \* handle* )

This function deinitializes the DSPI module and the related RTOS context.

Parameters

|               |                       |
|---------------|-----------------------|
| <i>handle</i> | The RTOS DSPI handle. |
|---------------|-----------------------|

### 11.5.3.3 status\_t DSPI\_RTOS\_Transfer ( *dspi\_rtos\_handle\_t \* handle, dspi\_transfer\_t \* transfer* )

This function performs an SPI transfer according to data given in the transfer structure.

Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>handle</i>   | The RTOS DSPI handle.                         |
| <i>transfer</i> | Structure specifying the transfer parameters. |

Returns

status of the operation.

## 11.6 DSPI µCOS/II Driver

### 11.6.1 Overview

#### Data Structures

- struct `dspi_rtos_handle_t`  
*DSPI FreeRTOS handle.* [More...](#)

#### DSPI RTOS Operation

- status\_t `DSPI_RTOS_Init` (`dspi_rtos_handle_t` \*handle, `SPI_Type` \*base, const `dspi_master_config_t` \*masterConfig, `uint32_t` srcClock\_Hz)  
*Initializes DSPI.*
- status\_t `DSPI_RTOS_Deinit` (`dspi_rtos_handle_t` \*handle)  
*Deinitializes the DSPI.*
- status\_t `DSPI_RTOS_Transfer` (`dspi_rtos_handle_t` \*handle, `dspi_transfer_t` \*transfer)  
*Performs SPI transfer.*

### 11.6.2 Data Structure Documentation

#### 11.6.2.1 struct `dspi_rtos_handle_t`

DSPI µC/OS-III handle.

DSPI µC/OS-II handle.

#### Data Fields

- `SPI_Type` \* `base`  
*DSPI base address.*
- `dspi_master_handle_t` `drv_handle`  
*Handle of the underlying driver, treated as opaque by the RTOS layer.*
- `SemaphoreHandle_t` `mutex`  
*Mutex to lock the handle during a transfer.*
- `SemaphoreHandle_t` `event`  
*Semaphore to notify and unblock task when transfer ends.*
- `OS_EVENT` \* `mutex`  
*Mutex to lock the handle during a transfer.*
- `OS_FLAG_GRP` \* `event`  
*Semaphore to notify and unblock task when transfer ends.*
- `OS_SEM` `mutex`  
*Mutex to lock the handle during a transfer.*
- `OS_FLAG_GRP` `event`  
*Semaphore to notify and unblock task when transfer ends.*

### 11.6.3 Function Documentation

11.6.3.1 `status_t DSPI_RTOS_Init( dspi_rtos_handle_t * handle, SPI_Type * base, const dspi_master_config_t * masterConfig, uint32_t srcClock_Hz )`

This function initializes the DSPI module and the related RTOS context.

Parameters

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

Returns

status of the operation.

### 11.6.3.2 status\_t DSPI\_RTOS\_Deinit ( *dspi\_rtos\_handle\_t \* handle* )

This function deinitializes the DSPI module and the related RTOS context.

Parameters

|               |                       |
|---------------|-----------------------|
| <i>handle</i> | The RTOS DSPI handle. |
|---------------|-----------------------|

### 11.6.3.3 status\_t DSPI\_RTOS\_Transfer ( *dspi\_rtos\_handle\_t \* handle, dspi\_transfer\_t \* transfer* )

This function performs an SPI transfer according to data given in the transfer structure.

Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>handle</i>   | The RTOS DSPI handle.                         |
| <i>transfer</i> | Structure specifying the transfer parameters. |

Returns

status of the operation.

### 11.7 DSPI µCOS/III Driver

#### 11.7.1 Overview

#### Data Structures

- struct `dspi_rtos_handle_t`  
*DSPI FreeRTOS handle.* [More...](#)

#### DSPI RTOS Operation

- status\_t `DSPI_RTOS_Init` (`dspi_rtos_handle_t` \*handle, `SPI_Type` \*base, const `dspi_master_config_t` \*masterConfig, `uint32_t` srcClock\_Hz)  
*Initializes DSPI.*
- status\_t `DSPI_RTOS_Deinit` (`dspi_rtos_handle_t` \*handle)  
*Deinitializes the DSPI.*
- status\_t `DSPI_RTOS_Transfer` (`dspi_rtos_handle_t` \*handle, `dspi_transfer_t` \*transfer)  
*Performs SPI transfer.*

#### 11.7.2 Data Structure Documentation

##### 11.7.2.1 struct `dspi_rtos_handle_t`

DSPI µC/OS-III handle.

DSPI µC/OS-II handle.

#### Data Fields

- `SPI_Type` \* `base`  
*DSPI base address.*
- `dspi_master_handle_t` `drv_handle`  
*Handle of the underlying driver, treated as opaque by the RTOS layer.*
- `SemaphoreHandle_t` `mutex`  
*Mutex to lock the handle during a transfer.*
- `SemaphoreHandle_t` `event`  
*Semaphore to notify and unblock task when transfer ends.*
- `OS_EVENT` \* `mutex`  
*Mutex to lock the handle during a transfer.*
- `OS_FLAG_GRP` \* `event`  
*Semaphore to notify and unblock task when transfer ends.*
- `OS_SEM` `mutex`  
*Mutex to lock the handle during a transfer.*
- `OS_FLAG_GRP` `event`  
*Semaphore to notify and unblock task when transfer ends.*

### 11.7.3 Function Documentation

**11.7.3.1 status\_t DSPI\_RTOS\_Init ( *dspi\_rtos\_handle\_t \* handle*, *SPI\_Type \* base*, *const dspi\_master\_config\_t \* masterConfig*, *uint32\_t srcClock\_Hz* )**

This function initializes the DSPI module and the related RTOS context.

## DSPI µCOS/III Driver

Parameters

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

Returns

status of the operation.

### 11.7.3.2 status\_t DSPI\_RTOS\_Deinit ( *dspi\_rtos\_handle\_t \* handle* )

This function deinitializes the DSPI module and the related RTOS context.

Parameters

|               |                       |
|---------------|-----------------------|
| <i>handle</i> | The RTOS DSPI handle. |
|---------------|-----------------------|

### 11.7.3.3 status\_t DSPI\_RTOS\_Transfer ( *dspi\_rtos\_handle\_t \* handle*, *dspi\_transfer\_t \* transfer* )

This function performs an SPI transfer according to data given in the transfer structure.

Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>handle</i>   | The RTOS DSPI handle.                         |
| <i>transfer</i> | Structure specifying the transfer parameters. |

Returns

status of the operation.

# Chapter 12

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

### 12.1 Overview

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

### 12.2 Typical use case

#### 12.2.1 eDMA Operation

```
edma_transfer_config_t transferConfig;
edma_config_t userConfig;
uint32_t transferDone = false;

EDMA_GetDefaultConfig(&userConfig);
EDMA_Init(DMA0, &userConfig);
EDMA_CreateHandle(&g_EDMA_Handle, DMA0, channel);
EDMA_SetCallback(&g_EDMA_Handle, EDMA_Callback, &transferDone);
EDMA_PrepTransfer(&transferConfig, srcAddr, srcWidth, destAddr, destWidth,
                  bytesEachRequest, transferBytes, kEDMA_MemoryToMemory);
EDMA_SubmitTransfer(&g_EDMA_Handle, &transferConfig, true);
EDMA_StartTransfer(&g_EDMA_Handle);
/* Wait for eDMA transfer finish */
while (transferDone != true);
```

## Data Structures

- struct `edma_config_t`  
*eDMA global configuration structure.* [More...](#)
- struct `edma_transfer_config_t`  
*eDMA transfer configuration* [More...](#)
- struct `edma_channel_Preemption_config_t`  
*eDMA channel priority configuration* [More...](#)
- struct `edma_minor_offset_config_t`  
*eDMA minor offset configuration* [More...](#)
- struct `edma_tcd_t`  
*eDMA TCD.* [More...](#)
- struct `edma_handle_t`  
*eDMA transfer handle structure* [More...](#)

## Macros

- #define `DMA_DCHPRI_INDEX`(channel) (((channel) & ~0x03U) | (3 - ((channel)&0x03U)))  
*Compute the offset unit from DCHPRI3.*
- #define `DMA_DCHPRIIn`(base, channel) ((volatile uint8\_t \*)(&(base->DCHPRI3)))[`DMA_DCHPRI_INDEX`(channel)]  
*Get the pointer of DCHPRIIn.*

## Typical use case

### TypeDefs

- `typedef void(* edma_callback )(struct _edma_handle *handle, void *userData, bool transferDone, uint32_t tcds)`  
*Define Callback function for eDMA.*

### Enumerations

- `enum edma_transfer_size_t {`  
 `kEDMA_TransferSize1Bytes = 0x0U,`  
 `kEDMA_TransferSize2Bytes = 0x1U,`  
 `kEDMA_TransferSize4Bytes = 0x2U,`  
 `kEDMA_TransferSize16Bytes = 0x4U,`  
 `kEDMA_TransferSize32Bytes = 0x5U }`  
*eDMA transfer configuration*
- `enum edma_modulo_t {`

```

kEDMA_ModuloDisable = 0x0U,
kEDMA_Modulo2bytes,
kEDMA_Modulo4bytes,
kEDMA_Modulo8bytes,
kEDMA_Modulo16bytes,
kEDMA_Modulo32bytes,
kEDMA_Modulo64bytes,
kEDMA_Modulo128bytes,
kEDMA_Modulo256bytes,
kEDMA_Modulo512bytes,
kEDMA_Modulo1Kbytes,
kEDMA_Modulo2Kbytes,
kEDMA_Modulo4Kbytes,
kEDMA_Modulo8Kbytes,
kEDMA_Modulo16Kbytes,
kEDMA_Modulo32Kbytes,
kEDMA_Modulo64Kbytes,
kEDMA_Modulo128Kbytes,
kEDMA_Modulo256Kbytes,
kEDMA_Modulo512Kbytes,
kEDMA_Modulo1Mbytes,
kEDMA_Modulo2Mbytes,
kEDMA_Modulo4Mbytes,
kEDMA_Modulo8Mbytes,
kEDMA_Modulo16Mbytes,
kEDMA_Modulo32Mbytes,
kEDMA_Modulo64Mbytes,
kEDMA_Modulo128Mbytes,
kEDMA_Modulo256Mbytes,
kEDMA_Modulo512Mbytes,
kEDMA_Modulo1Gbytes,
kEDMA_Modulo2Gbytes }

eDMA modulo configuration
• enum edma_bandwidth_t {
  kEDMA_BandwidthStallNone = 0x0U,
  kEDMA_BandwidthStall4Cycle = 0x2U,
  kEDMA_BandwidthStall8Cycle = 0x3U }

Bandwidth control.
• enum edma_channel_link_type_t {
  kEDMA_LinkNone = 0x0U,
  kEDMA_MinorLink,
  kEDMA_MajorLink }

Channel link type.
• enum _edma_channel_status_flags {

```

## Typical use case

- ```
kEDMA_DoneFlag = 0x1U,  
kEDMA_ErrorFlag = 0x2U,  
kEDMA_InterruptFlag = 0x4U }  
    eDMA channel status flags.  
• enum _edma_error_status_flags {  
    kEDMA_DestinationBusErrorFlag = DMA_ES_DBE_MASK,  
    kEDMA_SourceBusErrorFlag = DMA_ES_SBE_MASK,  
    kEDMA_ScatterGatherErrorFlag = DMA_ES_SGE_MASK,  
    kEDMA_NbytesErrorFlag = DMA_ES_NCE_MASK,  
    kEDMA_DestinationOffsetErrorFlag = DMA_ES_DOE_MASK,  
    kEDMA_DestinationAddressErrorFlag = DMA_ES_DAE_MASK,  
    kEDMA_SourceOffsetErrorFlag = DMA_ES_SOE_MASK,  
    kEDMA_SourceAddressErrorFlag = DMA_ES_SAE_MASK,  
    kEDMA_ErrorChannelFlag = DMA_ES_ERRCHN_MASK,  
    kEDMA_ChannelPriorityErrorFlag = DMA_ES_CPE_MASK,  
    kEDMA_TransferCanceledFlag = DMA_ES_ECX_MASK,  
    kEDMA_ValidFlag = DMA_ES_VLD_MASK }  
    eDMA channel error status flags.  
• enum edma_interrupt_enable_t {  
    kEDMA_ErrorInterruptEnable = 0x1U,  
    kEDMA_MajorInterruptEnable = DMA_CSR_INTMAJOR_MASK,  
    kEDMA_HalfInterruptEnable = DMA_CSR_INTHALF_MASK }  
    eDMA interrupt source  
• enum edma_transfer_type_t {  
    kEDMA_MemoryToMemory = 0x0U,  
    kEDMA_PeripheralToMemory,  
    kEDMA_MemoryToPeripheral }  
    eDMA transfer type  
• enum _edma_transfer_status {  
    kStatus_EDMA_QueueFull = MAKE_STATUS(kStatusGroup_EDMA, 0),  
    kStatus_EDMA_Busy = MAKE_STATUS(kStatusGroup_EDMA, 1) }  
    eDMA transfer status
```

## Driver version

- #define **FSL\_EDMA\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 1))  
*eDMA driver version*

## eDMA initialization and De-initialization

- void **EDMA\_Init** (DMA\_Type \*base, const **edma\_config\_t** \*config)  
*Initializes eDMA peripheral.*
- void **EDMA\_Deinit** (DMA\_Type \*base)  
*Deinitializes eDMA peripheral.*
- void **EDMA\_GetDefaultConfig** (**edma\_config\_t** \*config)  
*Gets the eDMA default configuration structure.*

## eDMA Channel Operation

- void [EDMA\\_ResetChannel](#) (DMA\_Type \*base, uint32\_t channel)  
*Sets all TCD registers to a default value.*
- void [EDMA\\_SetTransferConfig](#) (DMA\_Type \*base, uint32\_t channel, const [edma\\_transfer\\_config\\_t](#) \*config, [edma\\_tcd\\_t](#) \*nextTcd)  
*Configures the eDMA transfer attribute.*
- void [EDMA\\_SetMinorOffsetConfig](#) (DMA\_Type \*base, uint32\_t channel, const [edma\\_minor\\_offset\\_config\\_t](#) \*config)  
*Configures the eDMA minor offset feature.*
- static void [EDMA\\_SetChannelPreemptionConfig](#) (DMA\_Type \*base, uint32\_t channel, const [edma\\_channel\\_Preemption\\_config\\_t](#) \*config)  
*Configures the eDMA channel preemption feature.*
- void [EDMA\\_SetChannelLink](#) (DMA\_Type \*base, uint32\_t channel, [edma\\_channel\\_link\\_type\\_t](#) type, uint32\_t linkedChannel)  
*Sets the channel link for the eDMA transfer.*
- void [EDMA\\_SetBandWidth](#) (DMA\_Type \*base, uint32\_t channel, [edma\\_bandwidth\\_t](#) bandWidth)  
*Sets the bandwidth for the eDMA transfer.*
- void [EDMA\\_SetModulo](#) (DMA\_Type \*base, uint32\_t channel, [edma\\_modulo\\_t](#) srcModulo, [edma\\_modulo\\_t](#) destModulo)  
*Sets the source modulo and destination modulo for eDMA transfer.*
- static void [EDMA\\_EnableAutoStopRequest](#) (DMA\_Type \*base, uint32\_t channel, bool enable)  
*Enables an auto stop request for the eDMA transfer.*
- void [EDMA\\_EnableChannelInterrupts](#) (DMA\_Type \*base, uint32\_t channel, uint32\_t mask)  
*Enables the interrupt source for the eDMA transfer.*
- void [EDMA\\_DisableChannelInterrupts](#) (DMA\_Type \*base, uint32\_t channel, uint32\_t mask)  
*Disables the interrupt source for the eDMA transfer.*

## eDMA TCD Operation

- void [EDMA\\_TcdReset](#) ([edma\\_tcd\\_t](#) \*tcd)  
*Sets all fields to default values for the TCD structure.*
- void [EDMA\\_TcdSetTransferConfig](#) ([edma\\_tcd\\_t](#) \*tcd, const [edma\\_transfer\\_config\\_t](#) \*config, [edma\\_tcd\\_t](#) \*nextTcd)  
*Configures the eDMA TCD transfer attribute.*
- void [EDMA\\_TcdSetMinorOffsetConfig](#) ([edma\\_tcd\\_t](#) \*tcd, const [edma\\_minor\\_offset\\_config\\_t](#) \*config)  
*Configures the eDMA TCD minor offset feature.*
- void [EDMA\\_TcdSetChannelLink](#) ([edma\\_tcd\\_t](#) \*tcd, [edma\\_channel\\_link\\_type\\_t](#) type, uint32\_t linkedChannel)  
*Sets the channel link for eDMA TCD.*
- static void [EDMA\\_TcdSetBandWidth](#) ([edma\\_tcd\\_t](#) \*tcd, [edma\\_bandwidth\\_t](#) bandWidth)  
*Sets the bandwidth for the eDMA TCD.*
- void [EDMA\\_TcdSetModulo](#) ([edma\\_tcd\\_t](#) \*tcd, [edma\\_modulo\\_t](#) srcModulo, [edma\\_modulo\\_t](#) destModulo)  
*Sets the source modulo and destination modulo for eDMA TCD.*
- static void [EDMA\\_TcdEnableAutoStopRequest](#) ([edma\\_tcd\\_t](#) \*tcd, bool enable)  
*Enables the auto stop request for the eDMA TCD.*
- void [EDMA\\_TcdEnableInterrupts](#) ([edma\\_tcd\\_t](#) \*tcd, uint32\_t mask)  
*Enables the interrupt source for the eDMA TCD.*

## Typical use case

- void [EDMA\\_TcdDisableInterrupts](#) (`edma_tcd_t` \*tcd, `uint32_t` mask)  
*Disables the interrupt source for the eDMA TCD.*

## eDMA Channel Transfer Operation

- static void [EDMA\\_EnableChannelRequest](#) (`DMA_Type` \*base, `uint32_t` channel)  
*Enables the eDMA hardware channel request.*
- static void [EDMA\\_DisableChannelRequest](#) (`DMA_Type` \*base, `uint32_t` channel)  
*Disables the eDMA hardware channel request.*
- static void [EDMA\\_TriggerChannelStart](#) (`DMA_Type` \*base, `uint32_t` channel)  
*Starts the eDMA transfer by software trigger.*

## eDMA Channel Status Operation

- `uint32_t` [EDMA\\_GetRemainingBytes](#) (`DMA_Type` \*base, `uint32_t` channel)  
*Gets the Remaining bytes from the eDMA current channel TCD.*
- static `uint32_t` [EDMA\\_GetErrorStatusFlags](#) (`DMA_Type` \*base)  
*Gets the eDMA channel error status flags.*
- `uint32_t` [EDMA\\_GetChannelStatusFlags](#) (`DMA_Type` \*base, `uint32_t` channel)  
*Gets the eDMA channel status flags.*
- void [EDMA\\_ClearChannelStatusFlags](#) (`DMA_Type` \*base, `uint32_t` channel, `uint32_t` mask)  
*Clears the eDMA channel status flags.*

## eDMA Transactional Operation

- void [EDMA\\_CreateHandle](#) (`edma_handle_t` \*handle, `DMA_Type` \*base, `uint32_t` channel)  
*Creates the eDMA handle.*
- void [EDMA\\_InstallTCDMemory](#) (`edma_handle_t` \*handle, `edma_tcd_t` \*tcdPool, `uint32_t` tcdSize)  
*Installs the TCDs memory pool into eDMA handle.*
- void [EDMA\\_SetCallback](#) (`edma_handle_t` \*handle, `edma_callback` callback, `void` \*userData)  
*Installs a callback function for the eDMA transfer.*
- void [EDMA\\_PrepTransfer](#) (`edma_transfer_config_t` \*config, `void` \*srcAddr, `uint32_t` srcWidth, `void` \*destAddr, `uint32_t` destWidth, `uint32_t` bytesEachRequest, `uint32_t` transferBytes, `edma_transfer_type_t` type)  
*Prepares the eDMA transfer structure.*
- status\_t [EDMA\\_SubmitTransfer](#) (`edma_handle_t` \*handle, const `edma_transfer_config_t` \*config)  
*Submits the eDMA transfer request.*
- void [EDMA\\_StartTransfer](#) (`edma_handle_t` \*handle)  
*eDMA start transfer.*
- void [EDMA\\_StopTransfer](#) (`edma_handle_t` \*handle)  
*eDMA stop transfer.*
- void [EDMA\\_AbortTransfer](#) (`edma_handle_t` \*handle)  
*eDMA abort transfer.*
- void [EDMA\\_HandleIRQ](#) (`edma_handle_t` \*handle)  
*eDMA IRQ handler for current major loop transfer complete.*

## 12.3 Data Structure Documentation

### **12.3.1 struct edma\_config\_t**

## Data Fields

- bool `enableContinuousLinkMode`  
*Enable (true) continuous link mode.*
  - bool `enableHaltOnError`  
*Enable (true) transfer halt on error.*
  - bool `enableRoundRobinArbitration`  
*Enable (true) round robin channel arbitration method, or fixed priority arbitration is used for channel selection.*
  - bool `enableDebugMode`  
*Enable(true) eDMA debug mode.*

## **12.3.1.0.0.27 Field Documentation**

**12.3.1.0.0.27.1 bool edma\_config\_t::enableContinuousLinkMode**

Upon minor loop completion, the channel activates again if that channel has a minor loop channel link enabled and the link channel is itself.

**12.3.1.0.0.27.2 bool edma\_config\_t::enableHaltOnError**

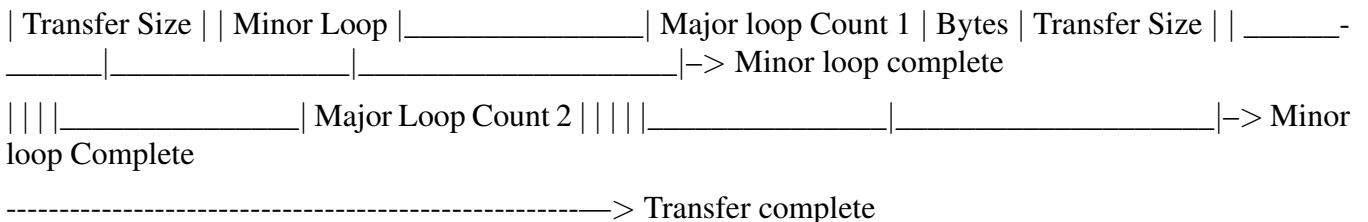
Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

### **12.3.1.0.0.27.3 bool edma\_config\_t::enableDebugMode**

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

### 12.3.2 struct edma\_transfer\_config\_t

This structure configures the source/destination transfer attribute. This figure shows the eDMA's transfer model:



## Data Structure Documentation

### Data Fields

- `uint32_t srcAddr`  
*Source data address.*
- `uint32_t destAddr`  
*Destination data address.*
- `edma_transfer_size_t srcTransferSize`  
*Source data transfer size.*
- `edma_transfer_size_t destTransferSize`  
*Destination data transfer size.*
- `int16_t srcOffset`  
*Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.*
- `int16_t destOffset`  
*Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.*
- `uint16_t minorLoopBytes`  
*Bytes to transfer in a minor loop.*
- `uint32_t majorLoopCounts`  
*Major loop iteration count.*

#### 12.3.2.0.0.28 Field Documentation

12.3.2.0.0.28.1 `uint32_t edma_transfer_config_t::srcAddr`

12.3.2.0.0.28.2 `uint32_t edma_transfer_config_t::destAddr`

12.3.2.0.0.28.3 `edma_transfer_size_t edma_transfer_config_t::srcTransferSize`

12.3.2.0.0.28.4 `edma_transfer_size_t edma_transfer_config_t::destTransferSize`

12.3.2.0.0.28.5 `int16_t edma_transfer_config_t::srcOffset`

12.3.2.0.0.28.6 `int16_t edma_transfer_config_t::destOffset`

12.3.2.0.0.28.7 `uint32_t edma_transfer_config_t::majorLoopCounts`

### 12.3.3 struct `edma_channel_Preemption_config_t`

### Data Fields

- `bool enableChannelPreemption`  
*If true: channel can be suspended by other channel with higher priority.*
- `bool enablePreemptAbility`  
*If true: channel can suspend other channel with low priority.*
- `uint8_t channelPriority`  
*Channel priority.*

### 12.3.4 struct edma\_minor\_offset\_config\_t

#### Data Fields

- bool **enableSrcMinorOffset**  
*Enable(true) or Disable(false) source minor loop offset.*
- bool **enableDestMinorOffset**  
*Enable(true) or Disable(false) destination minor loop offset.*
- uint32\_t **minorOffset**  
*Offset for minor loop mapping.*

#### 12.3.4.0.0.29 Field Documentation

12.3.4.0.0.29.1 bool **edma\_minor\_offset\_config\_t::enableSrcMinorOffset**

12.3.4.0.0.29.2 bool **edma\_minor\_offset\_config\_t::enableDestMinorOffset**

12.3.4.0.0.29.3 uint32\_t **edma\_minor\_offset\_config\_t::minorOffset**

### 12.3.5 struct edma\_tcd\_t

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

#### Data Fields

- \_\_IO uint32\_t **SADDR**  
*SADDR register, used to save source address.*
- \_\_IO uint16\_t **SOFF**  
*SOFF register, save offset bytes every transfer.*
- \_\_IO uint16\_t **ATTR**  
*ATTR register, source/destination transfer size and modulo.*
- \_\_IO uint32\_t **NBYTES**  
*Nbytes register, minor loop length in bytes.*
- \_\_IO uint32\_t **SLAST**  
*SLAST register.*
- \_\_IO uint32\_t **DADDR**  
*DADDR register, used for destination address.*
- \_\_IO uint16\_t **DOFF**  
*DOFF register, used for destination offset.*
- \_\_IO uint16\_t **CITER**  
*CITER register, current minor loop numbers, for unfinished minor loop.*
- \_\_IO uint32\_t **DLAST\_SGA**  
*DLASTSGA register, next stcd address used in scatter-gather mode.*
- \_\_IO uint16\_t **CSR**  
*CSR register, for TCD control status.*
- \_\_IO uint16\_t **BITER**  
*BITER register, begin minor loop count.*

## Data Structure Documentation

### 12.3.5.0.0.30 Field Documentation

12.3.5.0.0.30.1 `__IO uint16_t edma_tcd_t::CITER`

12.3.5.0.0.30.2 `__IO uint16_t edma_tcd_t::BITER`

### 12.3.6 struct `edma_handle_t`

#### Data Fields

- `edma_callback callback`  
*Callback function for major count exhausted.*
- `void * userData`  
*Callback function parameter.*
- `DMA_Type * base`  
*eDMA peripheral base address.*
- `edma_tcd_t * tcdPool`  
*Pointer to memory stored TCDs.*
- `uint8_t channel`  
*eDMA channel number.*
- `volatile int8_t header`  
*The first TCD index.*
- `volatile int8_t tail`  
*The last TCD index.*
- `volatile int8_t tcdUsed`  
*The number of used TCD slots.*
- `volatile int8_t tcdSize`  
*The total number of TCD slots in the queue.*
- `uint8_t flags`  
*The status of the current channel.*

**12.3.6.0.0.31 Field Documentation****12.3.6.0.0.31.1 `edma_callback` `edma_handle_t::callback`****12.3.6.0.0.31.2 `void*` `edma_handle_t::userData`****12.3.6.0.0.31.3 `DMA_Type*` `edma_handle_t::base`****12.3.6.0.0.31.4 `edma_tcd_t*` `edma_handle_t::tcdPool`****12.3.6.0.0.31.5 `uint8_t` `edma_handle_t::channel`****12.3.6.0.0.31.6 `volatile int8_t` `edma_handle_t::header`****12.3.6.0.0.31.7 `volatile int8_t` `edma_handle_t::tail`****12.3.6.0.0.31.8 `volatile int8_t` `edma_handle_t::tcdUsed`****12.3.6.0.0.31.9 `volatile int8_t` `edma_handle_t::tcdSize`****12.3.6.0.0.31.10 `uint8_t` `edma_handle_t::flags`****12.4 Macro Definition Documentation****12.4.1 `#define FSL_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))`**

Version 2.0.1.

**12.5 Typedef Documentation****12.5.1 `typedef void(* edma_callback)(struct _edma_handle *handle, void *userData, bool transferDone, uint32_t tcds)`****12.6 Enumeration Type Documentation****12.6.1 `enum edma_transfer_size_t`**

Enumerator

- kEDMA\_TransferSize1Bytes* Source/Destination data transfer size is 1 byte every time.
- kEDMA\_TransferSize2Bytes* Source/Destination data transfer size is 2 bytes every time.
- kEDMA\_TransferSize4Bytes* Source/Destination data transfer size is 4 bytes every time.
- kEDMA\_TransferSize16Bytes* Source/Destination data transfer size is 16 bytes every time.
- kEDMA\_TransferSize32Bytes* Source/Destination data transfer size is 32 bytes every time.

## Enumeration Type Documentation

### 12.6.2 enum edma\_modulo\_t

Enumerator

<i>kEDMA_ModuloDisable</i>	Disable modulo.
<i>kEDMA_Modulo2bytes</i>	Circular buffer size is 2 bytes.
<i>kEDMA_Modulo4bytes</i>	Circular buffer size is 4 bytes.
<i>kEDMA_Modulo8bytes</i>	Circular buffer size is 8 bytes.
<i>kEDMA_Modulo16bytes</i>	Circular buffer size is 16 bytes.
<i>kEDMA_Modulo32bytes</i>	Circular buffer size is 32 bytes.
<i>kEDMA_Modulo64bytes</i>	Circular buffer size is 64 bytes.
<i>kEDMA_Modulo128bytes</i>	Circular buffer size is 128 bytes.
<i>kEDMA_Modulo256bytes</i>	Circular buffer size is 256 bytes.
<i>kEDMA_Modulo512bytes</i>	Circular buffer size is 512 bytes.
<i>kEDMA_Modulo1Kbytes</i>	Circular buffer size is 1K bytes.
<i>kEDMA_Modulo2Kbytes</i>	Circular buffer size is 2K bytes.
<i>kEDMA_Modulo4Kbytes</i>	Circular buffer size is 4K bytes.
<i>kEDMA_Modulo8Kbytes</i>	Circular buffer size is 8K bytes.
<i>kEDMA_Modulo16Kbytes</i>	Circular buffer size is 16K bytes.
<i>kEDMA_Modulo32Kbytes</i>	Circular buffer size is 32K bytes.
<i>kEDMA_Modulo64Kbytes</i>	Circular buffer size is 64K bytes.
<i>kEDMA_Modulo128Kbytes</i>	Circular buffer size is 128K bytes.
<i>kEDMA_Modulo256Kbytes</i>	Circular buffer size is 256K bytes.
<i>kEDMA_Modulo512Kbytes</i>	Circular buffer size is 512K bytes.
<i>kEDMA_Modulo1Mbytes</i>	Circular buffer size is 1M bytes.
<i>kEDMA_Modulo2Mbytes</i>	Circular buffer size is 2M bytes.
<i>kEDMA_Modulo4Mbytes</i>	Circular buffer size is 4M bytes.
<i>kEDMA_Modulo8Mbytes</i>	Circular buffer size is 8M bytes.
<i>kEDMA_Modulo16Mbytes</i>	Circular buffer size is 16M bytes.
<i>kEDMA_Modulo32Mbytes</i>	Circular buffer size is 32M bytes.
<i>kEDMA_Modulo64Mbytes</i>	Circular buffer size is 64M bytes.
<i>kEDMA_Modulo128Mbytes</i>	Circular buffer size is 128M bytes.
<i>kEDMA_Modulo256Mbytes</i>	Circular buffer size is 256M bytes.
<i>kEDMA_Modulo512Mbytes</i>	Circular buffer size is 512M bytes.
<i>kEDMA_Modulo1Gbytes</i>	Circular buffer size is 1G bytes.
<i>kEDMA_Modulo2Gbytes</i>	Circular buffer size is 2G bytes.

### 12.6.3 enum edma\_bandwidth\_t

Enumerator

<i>kEDMA_BandwidthStallNone</i>	No eDMA engine stalls.
<i>kEDMA_BandwidthStall4Cycle</i>	eDMA engine stalls for 4 cycles after each read/write.
<i>kEDMA_BandwidthStall8Cycle</i>	eDMA engine stalls for 8 cycles after each read/write.

## 12.6.4 enum edma\_channel\_link\_type\_t

Enumerator

*kEDMA\_LinkNone* No channel link.

*kEDMA\_MinorLink* Channel link after each minor loop.

*kEDMA\_MajorLink* Channel link while major loop count exhausted.

## 12.6.5 enum \_edma\_channel\_status\_flags

Enumerator

*kEDMA\_DoneFlag* DONE flag, set while transfer finished, CITER value exhausted.

*kEDMA\_ErrorFlag* eDMA error flag, an error occurred in a transfer

*kEDMA\_InterruptFlag* eDMA interrupt flag, set while an interrupt occurred of this channel

## 12.6.6 enum \_edma\_error\_status\_flags

Enumerator

*kEDMA\_DestinationBusErrorFlag* Bus error on destination address.

*kEDMA\_SourceBusErrorFlag* Bus error on the source address.

*kEDMA\_ScatterGatherErrorFlag* Error on the Scatter/Gather address, not 32byte aligned.

*kEDMA\_NbytesErrorFlag* NBYTES/CITER configuration error.

*kEDMA\_DestinationOffsetErrorFlag* Destination offset not aligned with destination size.

*kEDMA\_DestinationAddressErrorFlag* Destination address not aligned with destination size.

*kEDMA\_SourceOffsetErrorFlag* Source offset not aligned with source size.

*kEDMA\_SourceAddressErrorFlag* Source address not aligned with source size.

*kEDMA\_ErrorChannelFlag* Error channel number of the cancelled channel number.

*kEDMA\_ChannelPriorityErrorFlag* Channel priority is not unique.

*kEDMA\_TransferCanceledFlag* Transfer cancelled.

*kEDMA\_ValidFlag* No error occurred, this bit is 0. Otherwise, it is 1.

## 12.6.7 enum edma\_interrupt\_enable\_t

Enumerator

*kEDMA\_ErrorInterruptEnable* Enable interrupt while channel error occurs.

*kEDMA\_MajorInterruptEnable* Enable interrupt while major count exhausted.

*kEDMA\_HalfInterruptEnable* Enable interrupt while major count to half value.

## Function Documentation

### 12.6.8 enum edma\_transfer\_type\_t

Enumerator

*kEDMA\_MemoryToMemory* Transfer from memory to memory.

*kEDMA\_PeripheralToMemory* Transfer from peripheral to memory.

*kEDMA\_MemoryToPeripheral* Transfer from memory to peripheral.

### 12.6.9 enum \_edma\_transfer\_status

Enumerator

*kStatus\_EDMA\_QueueFull* TCD queue is full.

*kStatus\_EDMA\_Busy* Channel is busy and can't handle the transfer request.

## 12.7 Function Documentation

### 12.7.1 void EDMA\_Init ( DMA\_Type \* *base*, const edma\_config\_t \* *config* )

This function ungates the eDMA clock and configures the eDMA peripheral according to the configuration structure.

Parameters

<i>base</i>	eDMA peripheral base address.
<i>config</i>	Pointer to configuration structure, see "edma_config_t".

Note

This function enable the minor loop map feature.

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

This function gates the eDMA clock.

Parameters

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

### 12.7.3 void EDMA\_GetDefaultConfig ( edma\_config\_t \* *config* )

This function sets the configuration structure to a default value. The default configuration is set to the following value:

```
* config.enableContinuousLinkMode = false;
* config.enableHaltOnError = true;
* config.enableRoundRobinArbitration = false;
* config.enableDebugMode = false;
*
```

## Parameters

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

**12.7.4 void EDMA\_ResetChannel ( DMA\_Type \* *base*, uint32\_t *channel* )**

This function sets TCD registers for this channel to default value.

## Parameters

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

## Note

This function must not be called while the channel transfer is on-going, or it causes unpredictable results.

This function enables the auto stop request feature.

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

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

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

## Function Documentation

Parameters

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

Note

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

### **12.7.6 void EDMA\_SetMinorOffsetConfig ( DMA\_Type \* *base*, uint32\_t *channel*, const edma\_minor\_offset\_config\_t \* *config* )**

Minor offset means signed-extended value added to source address or destination address after each minor loop.

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>config</i>	Pointer to Minor offset configuration structure.

### **12.7.7 static void EDMA\_SetChannelPreemptionConfig ( DMA\_Type \* *base*, uint32\_t *channel*, const edma\_channel\_Preemption\_config\_t \* *config* ) [inline], [static]**

This function configures the channel preemption attribute and the priority of the channel.

Parameters

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

<i>channel</i>	eDMA channel number
<i>config</i>	Pointer to channel preemption configuration structure.

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

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

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>type</i>	Channel link type, it can be one of: <ul style="list-style-type: none"><li>• kEDMA_LinkNone</li><li>• kEDMA_MinorLink</li><li>• kEDMA_MajorLink</li></ul>
<i>linkedChannel</i>	The linked channel number.

Note

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

### 12.7.9 void EDMA\_SetBandWidth ( DMA\_Type \* *base*, uint32\_t *channel*, edma\_bandwidth\_t *bandWidth* )

In general, because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

## Function Documentation

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>bandWidth</i>	Bandwidth setting, it can be one of: <ul style="list-style-type: none"><li>• kEDMABandwidthStallNone</li><li>• kEDMABandwidthStall4Cycle</li><li>• kEDMABandwidthStall8Cycle</li></ul>

### **12.7.10 void EDMA\_SetModulo ( DMA\_Type \* *base*, uint32\_t *channel*, edma\_modulo\_t *srcModulo*, edma\_modulo\_t *destModulo* )**

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>srcModulo</i>	Source modulo value.
<i>destModulo</i>	Destination modulo value.

### **12.7.11 static void EDMA\_EnableAutoStopRequest ( DMA\_Type \* *base*, uint32\_t *channel*, bool *enable* ) [inline], [static]**

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>enable</i>	The command for enable (true) or disable (false).

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

Parameters

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

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

Parameters

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

### 12.7.14 void EDMA\_TcdReset ( edma\_tcd\_t \* *tcd* )

This function sets all fields for this TCD structure to default value.

Parameters

<i>tcd</i>	Pointer to the TCD structure.
------------	-------------------------------

Note

This function enables the auto stop request feature.

### 12.7.15 void EDMA\_TcdSetTransferConfig ( edma\_tcd\_t \* *tcd*, const edma\_transfer\_config\_t \* *config*, edma\_tcd\_t \* *nextTcd* )

TCD is a transfer control descriptor. The content of the TCD is the same as hardware TCD registers. ST-TCD is used in scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```
*     edma_transfer_t config = {
*     ...
*     }
*     edma_tcd_t tcd __aligned(32);
```

## Function Documentation

```
*     edma_tcd_t nextTcd __aligned(32);
*     EDMA_TcdSetTransferConfig(&tcd, &config, &nextTcd);
*
```

### Parameters

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

### Note

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

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

### **12.7.16 void EDMA\_TcdSetMinorOffsetConfig ( *edma\_tcd\_t \* tcd, const edma\_minor\_offset\_config\_t \* config* )**

Minor offset is a signed-extended value added to the source address or destination address after each minor loop.

### Parameters

<i>tcd</i>	Point to the TCD structure.
<i>config</i>	Pointer to Minor offset configuration structure.

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

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

### Note

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

Parameters

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

### 12.7.18 static void EDMA\_TcdSetBandWidth ( *edma\_tcd\_t \* tcd*, *edma\_bandwidth\_t bandWidth* ) [inline], [static]

In general, because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. Bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

<i>tcd</i>	Point to the TCD structure.
<i>bandWidth</i>	Bandwidth setting, it can be one of: <ul style="list-style-type: none"><li>• kEDMABandwidthStallNone</li><li>• kEDMABandwidthStall4Cycle</li><li>• kEDMABandwidthStall8Cycle</li></ul>

### 12.7.19 void EDMA\_TcdSetModulo ( *edma\_tcd\_t \* tcd*, *edma\_modulo\_t srcModulo*, *edma\_modulo\_t destModulo* )

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

<i>tcd</i>	Point to the TCD structure.
------------	-----------------------------

## Function Documentation

<i>srcModulo</i>	Source modulo value.
<i>destModulo</i>	Destination modulo value.

### **12.7.20 static void EDMA\_TcdEnableAutoStopRequest ( edma\_tcd\_t \* *tcd*, bool *enable* ) [inline], [static]**

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

<i>tcd</i>	Point to the TCD structure.
<i>enable</i>	The command for enable(true) or disable(false).

### **12.7.21 void EDMA\_TcdEnableInterrupts ( edma\_tcd\_t \* *tcd*, uint32\_t *mask* )**

Parameters

<i>tcd</i>	Point to the TCD structure.
<i>mask</i>	The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type.

### **12.7.22 void EDMA\_TcdDisableInterrupts ( edma\_tcd\_t \* *tcd*, uint32\_t *mask* )**

Parameters

<i>tcd</i>	Point to the TCD structure.
<i>mask</i>	The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type.

### **12.7.23 static void EDMA\_EnableChannelRequest ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

This function enables the hardware channel request.

Parameters

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

#### **12.7.24 static void EDMA\_DisableChannelRequest ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

This function disables the hardware channel request.

Parameters

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

#### **12.7.25 static void EDMA\_TriggerChannelStart ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

This function starts a minor loop transfer.

Parameters

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

#### **12.7.26 uint32\_t EDMA\_GetRemainingBytes ( DMA\_Type \* *base*, uint32\_t *channel* )**

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

Parameters

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

## Function Documentation

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

Returns

Bytes have not been transferred yet for the current TCD.

Note

This function can only be used to get unfinished bytes of transfer without the next TCD, or it might be inaccuracy.

### **12.7.27 static uint32\_t EDMA\_GetErrorStatusFlags ( DMA\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The mask of error status flags. Users need to use the \_edma\_error\_status\_flags type to decode the return variables.

### **12.7.28 uint32\_t EDMA\_GetChannelStatusFlags ( DMA\_Type \* *base*, uint32\_t *channel* )**

Parameters

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

Returns

The mask of channel status flags. Users need to use the \_edma\_channel\_status\_flags type to decode the return variables.

### **12.7.29 void EDMA\_ClearChannelStatusFlags ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *mask* )**

Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>mask</i>	The mask of channel status to be cleared. Users need to use the defined _edma_channel_status_flags type.

### 12.7.30 void EDMA\_CreateHandle ( *edma\_handle\_t \* handle, DMA\_Type \* base, uint32\_t channel* )

This function is called if using transaction API for eDMA. This function initializes the internal state of eDMA handle.

Parameters

<i>handle</i>	eDMA handle pointer. The eDMA handle stores callback function and parameters.
<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.

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

This function is called after the EDMA\_CreateHandle to use scatter/gather feature.

Parameters

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

### 12.7.32 void EDMA\_SetCallback ( *edma\_handle\_t \* handle, edma\_callback callback, void \* userData* )

This callback is called in eDMA IRQ handler. Use the callback to do something after the current major loop transfer completes.

## Function Documentation

Parameters

<i>handle</i>	eDMA handle pointer.
<i>callback</i>	eDMA callback function pointer.
<i>userData</i>	Parameter for callback function.

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

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

Parameters

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

Note

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

**12.7.34 status\_t EDMA\_SubmitTransfer ( edma\_handle\_t \* *handle*, const edma\_transfer\_config\_t \* *config* )**

This function submits the eDMA transfer request according to the transfer configuration structure. If the user submits the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time.

Parameters

<i>handle</i>	eDMA handle pointer.
<i>config</i>	Pointer to eDMA transfer configuration structure.

Return values

<i>kStatus_EDMA_Success</i>	It means submit transfer request succeed.
<i>kStatus_EDMA_Queue-Full</i>	It means TCD queue is full. Submit transfer request is not allowed.
<i>kStatus_EDMA_Busy</i>	It means the given channel is busy, need to submit request later.

### 12.7.35 void EDMA\_StartTransfer ( *edma\_handle\_t \* handle* )

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

Parameters

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

### 12.7.36 void EDMA\_StopTransfer ( *edma\_handle\_t \* handle* )

This function disables the channel request to pause the transfer. Users can call [EDMA\\_StartTransfer\(\)](#) again to resume the transfer.

Parameters

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

### 12.7.37 void EDMA\_AbortTransfer ( *edma\_handle\_t \* handle* )

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

## Function Documentation

Parameters

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

### **12.7.38 void EDMA\_HandleIRQ ( *edma\_handle\_t* \* *handle* )**

This function clears the channel major interrupt flag and call the callback function if it is not NULL.

Parameters

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

# Chapter 13

## EWM: External Watchdog Monitor Driver

### 13.1 Overview

The KSDK provides a peripheral driver for the EWM module of Kinetis devices.

### 13.2 Typical use case

```
ewm_config_t config;
EWM_GetDefaultConfig(&config);
config.enableInterrupt = true;
config.compareLowValue = 0U;
config.compareHighValue = 0xAAU;
NVIC_EnableIRQ(WDOG_EWM_IRQn);
EWM_Init(base, &config);
```

## Data Structures

- struct `ewm_config_t`  
*Describes EWM clock source.* [More...](#)

## Enumerations

- enum `_ewm_interrupt_enable_t` { `kEWM_InterruptEnable` = EWM\_CTRL\_INTEN\_MASK }  
*EWM interrupt configuration structure, 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, 1))  
*EWM driver version 2.0.1.*

## 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)

## Enumeration Type Documentation

- static uint32\_t [EWM\\_GetStatusFlags](#) (EWM\_Type \*base)  
*Gets EWM all status flags.*
- void [EWM\\_Refresh](#) (EWM\_Type \*base)  
*Services the EWM.*

## 13.3 Data Structure Documentation

### 13.3.1 struct ewm\_config\_t

Data structure for EWM configuration.

This structure is used to configure the EWM.

#### Data Fields

- bool [enableEwm](#)  
*Enable EWM module.*
- bool [enableEwmInput](#)  
*Enable EWM\_in input.*
- bool [setInputAssertLogic](#)  
*EWM\_in signal assertion state.*
- bool [enableInterrupt](#)  
*Enable EWM interrupt.*
- uint8\_t [compareLowValue](#)  
*Compare low-register value.*
- uint8\_t [compareHighValue](#)  
*Compare high-register value.*

## 13.4 Macro Definition Documentation

### 13.4.1 #define FSL\_EWM\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

## 13.5 Enumeration Type Documentation

### 13.5.1 enum \_ewm\_interrupt\_enable\_t

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

Enumerator

**kEWM\_InterruptEnable** Enable EWM to generate an interrupt.

### 13.5.2 enum \_ewm\_status\_flags\_t

This structure contains the constants for the EWM status flags for use in the EWM functions.

Enumerator

**kEWM\_RunningFlag** Running flag, set when EWM is enabled.

## 13.6 Function Documentation

### 13.6.1 void EWM\_Init ( EWM\_Type \* *base*, const ewm\_config\_t \* *config* )

This function is used to initialize the EWM. After calling, the EWM runs immediately according to the configuration. Note that except for 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.

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 EWM

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

This function is used to shut down the EWM.

Parameters

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

### 13.6.3 void EWM\_GetDefaultConfig ( ewm\_config\_t \* *config* )

This function initializes the EWM configuration structure to default values. The default values are:

```
*     ewmConfig->enableEwm = true;
*     ewmConfig->enableEwmInput = false;
*     ewmConfig->setInputAssertLogic = false;
*     ewmConfig->enableInterrupt = false;
*     ewmConfig->ewm_lpo_clock_source_t = kEWM_LpoClockSource0;
*     ewmConfig->prescaler = 0;
*     ewmConfig->compareLowValue = 0;
*     ewmConfig->compareHighValue = 0xFEU;
*
```

## Function Documentation

Parameters

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

See Also

[ewm\\_config\\_t](#)

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

This function enables the EWM interrupt.

Parameters

<i>base</i>	EWM peripheral base address
<i>mask</i>	The interrupts to enable The parameter can be combination of the following source if defined: <ul style="list-style-type: none"><li>• kEWM InterruptEnable</li></ul>

### 13.6.5 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>

### 13.6.6 static **uint32\_t** EWM\_GetStatusFlags ( **EWM\_Type** \* *base* ) [**inline**], [**static**]

This function gets all status flags.

Example for getting Running Flag:

```
*     uint32_t status;
*     status = EWM_GetStatusFlags(ewm_base) & kEWM_RunningFlag;
*
```

## Parameters

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

## Returns

State of the status flag: asserted (true) or not-asserted (false).

## See Also

[\\_ewm\\_status\\_flags\\_t](#)

- true: a related status flag has been set.
- false: a related status flag is not set.

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

This function reset EWM counter to zero.

## Parameters

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

## Function Documentation

# Chapter 14

## C90TFS Flash Driver

### 14.1 Overview

The flash provides the C90TFS Flash driver of Kinetis devices with the C90TFS Flash module inside. The flash driver provides general APIs to handle specific operations on C90TFS/FTFx Flash module. The user can use those APIs directly in the application. In addition, it provides internal functions called by the driver. Although these functions are not meant to be called from the user's application directly, the APIs can still be used.

### Data Structures

- struct `flash_execute_in_ram_function_config_t`  
*Flash execute-in-RAM function information. [More...](#)*
- struct `flash_swap_state_config_t`  
*Flash Swap information. [More...](#)*
- struct `flash_swap_ifr_field_config_t`  
*Flash Swap IFR fields. [More...](#)*
- union `flash_swap_ifr_field_data_t`  
*Flash Swap IFR field data. [More...](#)*
- struct `flash_operation_config_t`  
*Active flash information for current operation. [More...](#)*
- struct `flash_config_t`  
*Flash driver state information. [More...](#)*

### Typedefs

- `typedef void(* flash_callback_t )(void)`  
*callback type used for pflash block*

### Enumerations

- enum `flash_margin_value_t` {  
  `kFLASH_MarginValueNormal`,  
  `kFLASH_MarginValueUser`,  
  `kFLASH_MarginValueFactory`,  
  `kFLASH_MarginValueInvalid` }  
*Enumeration for supported flash margin levels.*
- enum `flash_security_state_t` {  
  `kFLASH_SecurityStateNotSecure`,  
  `kFLASH_SecurityStateBackdoorEnabled`,  
  `kFLASH_SecurityStateBackdoorDisabled` }  
*Enumeration for the three possible flash security states.*

## Overview

- enum `flash_protection_state_t` {  
    `kFLASH_ProtectionStateUnprotected`,  
    `kFLASH_ProtectionStateProtected`,  
    `kFLASH_ProtectionStateMixed` }  
    *Enumeration for the three possible flash protection levels.*
- enum `flash_execute_only_access_state_t` {  
    `kFLASH_AccessStateUnLimited`,  
    `kFLASH_AccessStateExecuteOnly`,  
    `kFLASH_AccessStateMixed` }  
    *Enumeration for the three possible flash execute access levels.*
- enum `flash_property_tag_t` {  
    `kFLASH_PropertyPflashSectorSize` = 0x00U,  
    `kFLASH_PropertyPflashTotalSize` = 0x01U,  
    `kFLASH_PropertyPflashBlockSize` = 0x02U,  
    `kFLASH_PropertyPflashBlockCount` = 0x03U,  
    `kFLASH_PropertyPflashBlockBaseAddr` = 0x04U,  
    `kFLASH_PropertyPflashFacSupport` = 0x05U,  
    `kFLASH_PropertyPflashAccessSegmentSize` = 0x06U,  
    `kFLASH_PropertyPflashAccessSegmentCount` = 0x07U,  
    `kFLASH_PropertyFlexRamBlockBaseAddr` = 0x08U,  
    `kFLASH_PropertyFlexRamTotalSize` = 0x09U,  
    `kFLASH_PropertyDflashSectorSize` = 0x10U,  
    `kFLASH_PropertyDflashTotalSize` = 0x11U,  
    `kFLASH_PropertyDflashBlockSize` = 0x12U,  
    `kFLASH_PropertyDflashBlockCount` = 0x13U,  
    `kFLASH_PropertyDflashBlockBaseAddr` = 0x14U }  
    *Enumeration for various flash properties.*
- enum `_flash_execute_in_ram_function_constants` {  
    `kFLASH_ExecuteInRamFunctionMaxSizeInWords` = 16U,  
    `kFLASH_ExecuteInRamFunctionTotalNum` = 2U }  
    *Constants for execute-in-RAM flash function.*
- enum `flash_read_resource_option_t` {  
    `kFLASH_ResourceOptionFlashIfr`,  
    `kFLASH_ResourceOptionVersionId` = 0x01U }  
    *Enumeration for the two possible options of flash read resource command.*
- enum `_flash_read_resource_range` {  
    `kFLASH_ResourceRangePflashIfrSizeInBytes` = 256U,  
    `kFLASH_ResourceRangeVersionIdSizeInBytes` = 8U,  
    `kFLASH_ResourceRangeVersionIdStart` = 0x00U,  
    `kFLASH_ResourceRangeVersionIdEnd` = 0x07U ,  
    `kFLASH_ResourceRangePflashSwapIfrEnd`,  
    `kFLASH_ResourceRangeDflashIfrStart` = 0x800000U,  
    `kFLASH_ResourceRangeDflashIfrEnd` = 0x8003FFU }  
    *Enumeration for the range of special-purpose flash resource.*
- enum `flash_flexram_function_option_t` {  
    `kFLASH_FlexramFunctionOptionAvailableAsRam` = 0xFFU,

- **kFLASH\_FlexramFunctionOptionAvailableForEeprom** = 0x00U }
- Enumeration for the two possible options of set flexram function command.
- enum **\_flash\_acceleration\_ram\_property**
  - Enumeration for acceleration RAM property.
- enum **flash\_swap\_function\_option\_t** {
  - kFLASH\_SwapFunctionOptionEnable** = 0x00U,
  - kFLASH\_SwapFunctionOptionDisable** = 0x01U }
  - Enumeration for the possible options of Swap function.
- enum **flash\_swap\_control\_option\_t** {
  - kFLASH\_SwapControlOptionInitializeSystem** = 0x01U,
  - kFLASH\_SwapControlOptionSetInUpdateState** = 0x02U,
  - kFLASH\_SwapControlOptionSetInCompleteState** = 0x04U,
  - kFLASH\_SwapControlOptionReportStatus** = 0x08U,
  - kFLASH\_SwapControlOptionDisableSystem** = 0x10U }
  - Enumeration for the possible options of Swap Control commands.
- enum **flash\_swap\_state\_t** {
  - kFLASH\_SwapStateUninitialized** = 0x00U,
  - kFLASH\_SwapStateReady** = 0x01U,
  - kFLASH\_SwapStateUpdate** = 0x02U,
  - kFLASH\_SwapStateUpdateErased** = 0x03U,
  - kFLASH\_SwapStateComplete** = 0x04U,
  - kFLASH\_SwapStateDisabled** = 0x05U }
  - Enumeration for the possible flash swap status.
- enum **flash\_swap\_block\_status\_t** {
  - kFLASH\_SwapBlockStatusLowerHalfProgramBlocksAtZero**,
  - kFLASH\_SwapBlockStatusUpperHalfProgramBlocksAtZero** }
  - Enumeration for the possible flash swap block status
- enum **flash\_partition\_flexram\_load\_option\_t** {
  - kFLASH\_PartitionFlexramLoadOptionLoadedWithValidEepromData**,
  - kFLASH\_PartitionFlexramLoadOptionNotLoaded** = 0x01U }
  - Enumeration for FlexRAM load during reset option.

## Flash version

- enum **\_flash\_driver\_version\_constants** {
  - kFLASH\_DriverVersionName** = 'F',
  - kFLASH\_DriverVersionMajor** = 2,
  - kFLASH\_DriverVersionMinor** = 1,
  - kFLASH\_DriverVersionBugfix** = 0 }
  - FLASH driver version for ROM.*
- #define **MAKE\_VERSION**(major, minor, bugfix) (((major) << 16) | ((minor) << 8) | (bugfix))
  - Construct the version number for drivers.*
- #define **FSL\_FLASH\_DRIVER\_VERSION** (**MAKE\_VERSION**(2, 1, 0))
  - FLASH driver version for SDK.*

## Flash configuration

- #define **FLASH\_SSD\_CONFIG\_ENABLE\_FLEXNVM\_SUPPORT** 1

## Overview

- *Whether to support FlexNVM in flash driver.*  
• `#define FLASH_SSD_IS_FLEXNVM_ENABLED (FLASH_SSD_CONFIG_ENABLE_FLEXN-VM_SUPPORT && FSL_FEATURE_FLASH_HAS_FLEX_NVM)`  
*Whether the FlexNVM is enabled in flash driver.*
- `#define FLASH_DRIVER_IS_FLASH_RESIDENT 1`  
*Flash driver location.*
- `#define FLASH_DRIVER_IS_EXPORTED 0`  
*Flash Driver Export option.*

## Flash status

- `enum _flash_status {`  
 `kStatus_FLASH_Success = MAKE_STATUS(kStatusGroupGeneric, 0),`  
 `kStatus_FLASH_InvalidArgument = MAKE_STATUS(kStatusGroupGeneric, 4),`  
 `kStatus_FLASH_SizeError = MAKE_STATUS(kStatusGroupFlashDriver, 0),`  
 `kStatus_FLASH_AlignmentError,`  
 `kStatus_FLASH_AddressError = MAKE_STATUS(kStatusGroupFlashDriver, 2),`  
 `kStatus_FLASH_AccessError,`  
 `kStatus_FLASH_ProtectionViolation,`  
 `kStatus_FLASH_CommandFailure,`  
 `kStatus_FLASH_UnknownProperty = MAKE_STATUS(kStatusGroupFlashDriver, 6),`  
 `kStatus_FLASH_EraseKeyError = MAKE_STATUS(kStatusGroupFlashDriver, 7),`  
 `kStatus_FLASH_RegionExecuteOnly = MAKE_STATUS(kStatusGroupFlashDriver, 8),`  
 `kStatus_FLASH_ExecuteInRamFunctionNotReady,`  
 `kStatus_FLASH_PartitionStatusUpdateFailure,`  
 `kStatus_FLASH_SetFlexramAsEepromError,`  
 `kStatus_FLASH_RecoverFlexramAsRamError,`  
 `kStatus_FLASH_SetFlexramAsRamError = MAKE_STATUS(kStatusGroupFlashDriver, 13),`  
 `kStatus_FLASH_RecoverFlexramAsEepromError,`  
 `kStatus_FLASH_CommandNotSupported = MAKE_STATUS(kStatusGroupFlashDriver, 15),`  
 `kStatus_FLASH_SwapSystemNotInUninitialized,`  
 `kStatus_FLASH_SwapIndicatorAddressError }`  
*Flash driver status codes.*
- `#define kStatusGroupGeneric 0`  
*Flash driver status group.*
- `#define kStatusGroupFlashDriver 1`
- `#define MAKE_STATUS(group, code) (((group)*100) + (code))`  
*Construct a status code value from a group and code number.*

## Flash API key

- `enum _flash_driver_api_keys { kFLASH_ApiEraseKey = FOUR_CHAR_CODE('k', 'f', 'e', 'k') }`  
*Enumeration for flash driver API keys.*
- `#define FOUR_CHAR_CODE(a, b, c, d) (((d) << 24) | ((c) << 16) | ((b) << 8) | ((a)))`  
*Construct the four char code for flash driver API key.*

## Initialization

- status\_t **FLASH\_Init** (*flash\_config\_t* \*config)  
*Initializes global flash properties structure members.*
- status\_t **FLASH\_SetCallback** (*flash\_config\_t* \*config, *flash\_callback\_t* callback)  
*Set the desired flash callback function.*
- status\_t **FLASH\_PrepareExecuteInRamFunctions** (*flash\_config\_t* \*config)  
*Prepare flash execute-in-RAM functions.*

## Erasing

- status\_t **FLASH\_EraseAll** (*flash\_config\_t* \*config, uint32\_t key)  
*Erases entire flash.*
- status\_t **FLASH\_Erase** (*flash\_config\_t* \*config, uint32\_t start, uint32\_t lengthInBytes, uint32\_t key)  
*Erases flash sectors encompassed by parameters passed into function.*
- status\_t **FLASH\_EraseAllExecuteOnlySegments** (*flash\_config\_t* \*config, uint32\_t key)  
*Erases entire flash, including protected sectors.*

## Programming

- status\_t **FLASH\_Program** (*flash\_config\_t* \*config, uint32\_t start, uint32\_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, uint32\_t \*src, uint32\_t lengthInBytes)  
*Programs Program Once Field through parameters.*

## Reading

Programs flash with data at locations passed in through parameters via Program Section command

This function programs the flash memory with desired data for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	Pointer to 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>	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.

## Overview

Return values

<i>kStatus_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FLASH_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FLASH_AddressError</i>	Address is out of range.
<i>kStatus_FLASH_SetFlexramAsRamError</i>	Failed to set flexram as RAM
<i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FLASH_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FLASH_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FLASH_CommandFailure</i>	Run-time error during command execution.
<i>kStatus_FLASH_RecoverFlexramAsEepromError</i>	Failed to recover flexram as eeprom

Programs EEPROM with data at locations passed in through parameters

This function programs the Emulated EEPROM with desired data for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	Pointer to 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>	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_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FLASH_AddressError</i>	Address is out of range.
<i>kStatus_FLASH_SetFlexramAsEepromError</i>	Failed to set flexram as eeprom.
<i>kStatus_FLASH_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FLASH_RecoverFlexramAsRamError</i>	Failed to recover flexram as RAM

- status\_t **FLASH\_ReadOnce** (*flash\_config\_t* \*config, uint32\_t index, uint32\_t \*dst, uint32\_t lengthInBytes)

*Read resource with data at locations passed in through parameters.*

## Security

- status\_t **FLASH\_GetSecurityState** (*flash\_config\_t* \*config, *flash\_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 user to bypass security with a backdoor key.*

## Verification

- status\_t **FLASH\_VerifyEraseAll** (*flash\_config\_t* \*config, *flash\_margin\_value\_t* margin)  
*Verifies erasure of entire flash at specified margin level.*
- status\_t **FLASH\_VerifyErase** (*flash\_config\_t* \*config, uint32\_t start, uint32\_t lengthInBytes, *flash\_margin\_value\_t* margin)  
*Verifies erasure of desired flash area at specified margin level.*
- status\_t **FLASH\_VerifyProgram** (*flash\_config\_t* \*config, uint32\_t start, uint32\_t lengthInBytes, const uint32\_t \*expectedData, *flash\_margin\_value\_t* margin, uint32\_t \*failedAddress, uint32\_t \*failedData)  
*Verifies programming of desired flash area at specified margin level.*
- status\_t **FLASH\_VerifyEraseAllExecuteOnlySegments** (*flash\_config\_t* \*config, *flash\_margin\_value\_t* margin)  
*Verifies if the program flash executeonly segments have been erased to the specified read margin level.*

## Protection

- status\_t **FLASH\_IsProtected** (*flash\_config\_t* \*config, uint32\_t start, uint32\_t lengthInBytes, *flash\_protection\_state\_t* \*protection\_state)  
*Returns the protection state of desired flash area via the pointer passed into the function.*
- status\_t **FLASH\_IsExecuteOnly** (*flash\_config\_t* \*config, uint32\_t start, uint32\_t lengthInBytes, *flash\_execute\_only\_access\_state\_t* \*access\_state)  
*Returns the access state of desired flash area via the pointer passed into the function.*

## Overview

## Properties

- status\_t [FLASH\\_GetProperty](#) (flash\_config\_t \*config, flash\_property\_tag\_t whichProperty, uint32\_t \*value)  
*Returns the desired flash property.*

## Flash Protection Utilities

Prepares the FlexNVM block for use as data flash, EEPROM backup, or a combination of both and initializes the FlexRAM.

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

<a href="#">kStatus_FLASH_Success</a>	API was executed successfully.
<a href="#">kStatus_FLASH_InvalidArgument</a>	Invalid argument is provided.
<a href="#">kStatus_FLASH_ExecuteInRamFunctionNotReady</a>	Execute-in-RAM function is not available.
<a href="#">kStatus_FLASH_AccessError</a>	Invalid instruction codes and out-of bounds addresses.
<a href="#">kStatus_FLASH_ProtectionViolation</a>	The program/erase operation is requested to execute on protected areas.
<a href="#">kStatus_FLASH_CommandFailure</a>	Run-time error during command execution.

- status\_t [FLASH\\_PflashSetProtection](#) (flash\_config\_t \*config, uint32\_t protectStatus)  
*Set PFLASH Protection to the intended protection status.*
- status\_t [FLASH\\_PflashGetProtection](#) (flash\_config\_t \*config, uint32\_t \*protectStatus)  
*Get PFLASH Protection Status.*

## 14.2 Data Structure Documentation

### 14.2.1 struct flash\_execute\_in\_ram\_function\_config\_t

#### Data Fields

- `uint32_t activeFunctionCount`  
*Number of available execute-in-RAM functions.*
- `uint32_t * flashRunCommand`  
*execute-in-RAM function: flash\_run\_command.*
- `uint32_t * flashCacheClearCommand`  
*execute-in-RAM function: flash\_cache\_clear\_command.*

#### 14.2.1.0.0.32 Field Documentation

##### 14.2.1.0.0.32.1 `uint32_t flash_execute_in_ram_function_config_t::activeFunctionCount`

##### 14.2.1.0.0.32.2 `uint32_t* flash_execute_in_ram_function_config_t::flashRunCommand`

##### 14.2.1.0.0.32.3 `uint32_t* flash_execute_in_ram_function_config_t::flashCacheClearCommand`

### 14.2.2 struct flash\_swap\_state\_config\_t

#### Data Fields

- `flash_swap_state_t flashSwapState`  
*Current swap system status.*
- `flash_swap_block_status_t currentSwapBlockStatus`  
*Current swap block status.*
- `flash_swap_block_status_t nextSwapBlockStatus`  
*Next swap block status.*

#### 14.2.2.0.0.33 Field Documentation

##### 14.2.2.0.0.33.1 `flash_swap_state_t flash_swap_state_config_t::flashSwapState`

##### 14.2.2.0.0.33.2 `flash_swap_block_status_t flash_swap_state_config_t::currentSwapBlockStatus`

##### 14.2.2.0.0.33.3 `flash_swap_block_status_t flash_swap_state_config_t::nextSwapBlockStatus`

### 14.2.3 struct flash\_swap\_ifr\_field\_config\_t

#### Data Fields

- `uint16_t swapIndicatorAddress`  
*Swap indicator address field.*
- `uint16_t swapEnableWord`  
*Swap enable word field.*
- `uint8_t reserved0 [4]`

## Data Structure Documentation

*Reserved field.*

### 14.2.3.0.0.34 Field Documentation

14.2.3.0.0.34.1 `uint16_t flash_swap_ifr_field_config_t::swapIndicatorAddress`

14.2.3.0.0.34.2 `uint16_t flash_swap_ifr_field_config_t::swapEnableWord`

14.2.3.0.0.34.3 `uint8_t flash_swap_ifr_field_config_t::reserved0[4]`

### 14.2.4 union flash\_swap\_ifr\_field\_data\_t

#### Data Fields

- `uint32_t flashSwapIfrData [2]`  
*Flash Swap IFR field data.*
- `flash_swap_ifr_field_config_t flashSwapIfrField`  
*Flash Swap IFR field struct.*

### 14.2.4.0.0.35 Field Documentation

14.2.4.0.0.35.1 `uint32_t flash_swap_ifr_field_data_t::flashSwapIfrData[2]`

14.2.4.0.0.35.2 `flash_swap_ifr_field_config_t flash_swap_ifr_field_data_t::flashSwapIfrField`

### 14.2.5 struct flash\_operation\_config\_t

#### Data Fields

- `uint32_t convertedAddress`  
*Converted address for current flash type.*
- `uint32_t activeSectorSize`  
*Sector size of current flash type.*
- `uint32_t activeBlockSize`  
*Block size of current flash type.*
- `uint32_t blockWriteUnitSize`  
*write unit size.*
- `uint32_t sectorCmdAddressAlignment`  
*Erase sector command address alignment.*
- `uint32_t partCmdAddressAlignment`  
*Program/Verify part command address alignment.*
- `32_t resourceCmdAddressAlignment`  
*Read resource command address alignment.*
- `uint32_t checkCmdAddressAlignment`  
*Program check command address alignment.*

#### 14.2.5.0.0.36 Field Documentation

- 14.2.5.0.0.36.1 `uint32_t flash_operation_config_t::convertedAddress`
- 14.2.5.0.0.36.2 `uint32_t flash_operation_config_t::activeSectorSize`
- 14.2.5.0.0.36.3 `uint32_t flash_operation_config_t::activeBlockSize`
- 14.2.5.0.0.36.4 `uint32_t flash_operation_config_t::blockWriteUnitSize`
- 14.2.5.0.0.36.5 `uint32_t flash_operation_config_t::sectorCmdAddressAlignment`
- 14.2.5.0.0.36.6 `uint32_t flash_operation_config_t::partCmdAddressAlignment`
- 14.2.5.0.0.36.7 `uint32_t flash_operation_config_t::resourceCmdAddressAlignment`
- 14.2.5.0.0.36.8 `uint32_t flash_operation_config_t::checkCmdAddressAlignment`

#### 14.2.6 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.

#### Data Fields

- `uint32_t PFlashBlockBase`  
*Base address of the first PFlash block.*
- `uint32_t PFlashTotalSize`  
*Size of all combined PFlash block.*
- `uint32_t PFlashBlockCount`  
*Number of PFlash blocks.*
- `uint32_t PFlashSectorSize`  
*Size in bytes of a sector of PFlash.*
- `flash_callback_t PFlashCallback`  
*Callback function for flash API.*
- `uint32_t PFlashAccessSegmentSize`  
*Size in bytes of a access segment of PFlash.*
- `uint32_t PFlashAccessSegmentCount`  
*Number of PFlash access segments.*
- `uint32_t * flashExecuteInRamFunctionInfo`  
*Info struct of flash execute-in-RAM function.*
- `uint32_t FlexRAMBlockBase`  
*For FlexNVM device, this is the base address of FlexRAM For non-FlexNVM device, this is the base address of acceleration RAM memory.*
- `uint32_t FlexRAMTotalSize`  
*For FlexNVM device, this is the size of FlexRAM For non-FlexNVM device, this is the size of acceleration RAM memory.*
- `uint32_t DFlashBlockBase`  
*For FlexNVM device, this is the base address of D-Flash memory (FlexNVM memory); For non-FlexNVM*

## Macro Definition Documentation

- `device, this field is unused.`  
• `uint32_t DFlashTotalSize`  
*For FlexNVM device, this is total size of the FlexNVM memory; For non-FlexNVM device, this field is unused.*  
• `uint32_t EEpromTotalSize`  
*For FlexNVM device, this is the size in byte of EEPROM area which was partitioned from FlexRAM; For non-FlexNVM device, this field is unused.*

### 14.2.6.0.0.37 Field Documentation

14.2.6.0.0.37.1 `uint32_t flash_config_t::PFlashTotalSize`

14.2.6.0.0.37.2 `uint32_t flash_config_t::PFlashBlockCount`

14.2.6.0.0.37.3 `uint32_t flash_config_t::PFlashSectorSize`

14.2.6.0.0.37.4 `flash_callback_t flash_config_t::PFlashCallback`

14.2.6.0.0.37.5 `uint32_t flash_config_t::PFlashAccessSegmentSize`

14.2.6.0.0.37.6 `uint32_t flash_config_t::PFlashAccessSegmentCount`

14.2.6.0.0.37.7 `uint32_t* flash_config_t::flashExecuteInRamFunctionInfo`

## 14.3 Macro Definition Documentation

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

14.3.2 `#define FSL_FLASH_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

Version 2.1.0.

14.3.3 `#define FLASH_SSD_CONFIG_ENABLE_FLEXNVM_SUPPORT 1`

Enable FlexNVM support by default.

14.3.4 `#define FLASH_DRIVER_IS_FLASH_RESIDENT 1`

Used for flash resident application.

14.3.5 `#define FLASH_DRIVER_IS_EXPORTED 0`

Used for SDK application.

**14.3.6 #define kStatusGroupGeneric 0**

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

**14.3.8 #define FOUR\_CHAR\_CODE( a, b, c, d ) (((d) << 24) | ((c) << 16) | ((b) << 8) | ((a)))**

## 14.4 Enumeration Type Documentation

### 14.4.1 enum \_flash\_driver\_version\_constants

Enumerator

***kFLASH\_DriverVersionName*** Flash driver version name.

***kFLASH\_DriverVersionMajor*** Major flash driver version.

***kFLASH\_DriverVersionMinor*** Minor flash driver version.

***kFLASH\_DriverVersionBugfix*** Bugfix for flash driver version.

### 14.4.2 enum \_flash\_status

Enumerator

***kStatus\_FLASH\_Success*** API is executed successfully.

***kStatus\_FLASH\_InvalidArgument*** Invalid argument.

***kStatus\_FLASH\_SizeError*** Error size.

***kStatus\_FLASH\_AlignmentError*** Parameter is not aligned with specified baseline.

***kStatus\_FLASH\_AddressError*** Address is out of range.

***kStatus\_FLASH\_AccessError*** Invalid instruction codes and out-of bounds addresses.

***kStatus\_FLASH\_ProtectionViolation*** The program/erase operation is requested to execute on protected areas.

***kStatus\_FLASH\_CommandFailure*** Run-time error during command execution.

***kStatus\_FLASH\_UnknownProperty*** Unknown property.

***kStatus\_FLASH\_EraseKeyError*** API erase key is invalid.

***kStatus\_FLASH\_RegionExecuteOnly*** Current region is execute only.

***kStatus\_FLASH\_ExecuteInRamFunctionNotReady*** Execute-in-RAM function is not available.

***kStatus\_FLASH\_PartitionStatusUpdateFailure*** Failed to update partition status.

***kStatus\_FLASH\_SetFlexramAsEepromError*** Failed to set flexram as eeprom.

***kStatus\_FLASH\_RecoverFlexramAsRamError*** Failed to recover flexram as RAM.

***kStatus\_FLASH\_SetFlexramAsRamError*** Failed to set flexram as RAM.

***kStatus\_FLASH\_RecoverFlexramAsEepromError*** Failed to recover flexram as eeprom.

***kStatus\_FLASH\_CommandNotSupported*** Flash API is not supported.

***kStatus\_FLASH\_SwapSystemNotInUninitialized*** Swap system is not in uninitialized state.

***kStatus\_FLASH\_SwapIndicatorAddressError*** Swap indicator address is invalid.

## Enumeration Type Documentation

### 14.4.3 enum \_flash\_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

***kFLASH\_ApiEraseKey*** Key value used to validate all flash erase APIs.

### 14.4.4 enum flash\_margin\_value\_t

Enumerator

***kFLASH\_MarginValueNormal*** Use the 'normal' read level for 1s.

***kFLASH\_MarginValueUser*** Apply the 'User' margin to the normal read-1 level.

***kFLASH\_MarginValueFactory*** Apply the 'Factory' margin to the normal read-1 level.

***kFLASH\_MarginValueInvalid*** Not real margin level, Used to determine the range of valid margin level.

### 14.4.5 enum flash\_security\_state\_t

Enumerator

***kFLASH\_SecurityStateNotSecure*** Flash is not secure.

***kFLASH\_SecurityStateBackdoorEnabled*** Flash backdoor is enabled.

***kFLASH\_SecurityStateBackdoorDisabled*** Flash backdoor is disabled.

### 14.4.6 enum flash\_protection\_state\_t

Enumerator

***kFLASH\_ProtectionStateUnprotected*** Flash region is not protected.

***kFLASH\_ProtectionStateProtected*** Flash region is protected.

***kFLASH\_ProtectionStateMixed*** Flash is mixed with protected and unprotected region.

### 14.4.7 enum flash\_execute\_only\_access\_state\_t

Enumerator

***kFLASH\_AccessStateUnLimited*** Flash region is unLimited.

***kFLASH\_AccessStateExecuteOnly*** Flash region is execute only.

***kFLASH\_AccessStateMixed*** Flash is mixed with unLimited and execute only region.

#### 14.4.8 enum flash\_property\_tag\_t

Enumerator

***kFLASH\_PropertyPflashSectorSize*** Pflash sector size property.

***kFLASH\_PropertyPflashTotalSize*** Pflash total size property.

***kFLASH\_PropertyPflashBlockSize*** Pflash block size property.

***kFLASH\_PropertyPflashBlockCount*** Pflash block count property.

***kFLASH\_PropertyPflashBlockBaseAddr*** Pflash block base address property.

***kFLASH\_PropertyPflashFacSupport*** Pflash fac support property.

***kFLASH\_PropertyPflashAccessSegmentSize*** Pflash access segment size property.

***kFLASH\_PropertyPflashAccessSegmentCount*** Pflash access segment count property.

***kFLASH\_PropertyFlexRamBlockBaseAddr*** FlexRam block base address property.

***kFLASH\_PropertyFlexRamTotalSize*** FlexRam total size property.

***kFLASH\_PropertyDflashSectorSize*** Dflash sector size property.

***kFLASH\_PropertyDflashTotalSize*** Dflash total size property.

***kFLASH\_PropertyDflashBlockSize*** Dflash block count property.

***kFLASH\_PropertyDflashBlockCount*** Dflash block base address property.

***kFLASH\_PropertyDflashBlockBaseAddr*** Eeprom total size property.

#### 14.4.9 enum \_flash\_execute\_in\_ram\_function\_constants

Enumerator

***kFLASH\_ExecuteInRamFunctionMaxSizeInWords*** Max size of execute-in-RAM function.

***kFLASH\_ExecuteInRamFunctionTotalNum*** Total number of execute-in-RAM functions.

#### 14.4.10 enum flash\_read\_resource\_option\_t

Enumerator

***kFLASH\_ResourceOptionFlashIfr*** Select code for Program flash 0 IFR, Program flash swap 0 IFR, Data flash 0 IFR.

***kFLASH\_ResourceOptionVersionId*** Select code for Version ID.

## Enumeration Type Documentation

### 14.4.11 enum \_flash\_read\_resource\_range

Enumerator

*kFLASH\_ResourceRangePflashIfrSizeInBytes* Pflash IFR size in byte.  
*kFLASH\_ResourceRangeVersionIdSizeInBytes* Version ID IFR size in byte.  
*kFLASH\_ResourceRangeVersionIdStart* Version ID IFR start address.  
*kFLASH\_ResourceRangeVersionIdEnd* Version ID IFR end address.  
*kFLASH\_ResourceRangePflashSwapIfrEnd* Pflash swap IFR end address.  
*kFLASH\_ResourceRangeDflashIfrStart* Dflash IFR start address.  
*kFLASH\_ResourceRangeDflashIfrEnd* Dflash IFR end address.

### 14.4.12 enum flash\_flexram\_function\_option\_t

Enumerator

*kFLASH\_FlexramFunctionOptionAvailableAsRam* Option used to make FlexRAM available as RAM.  
*kFLASH\_FlexramFunctionOptionAvailableForEeprom* Option used to make FlexRAM available for EEPROM.

### 14.4.13 enum flash\_swap\_function\_option\_t

Enumerator

*kFLASH\_SwapFunctionOptionEnable* Option used to enable Swap function.  
*kFLASH\_SwapFunctionOptionDisable* Option used to Disable Swap function.

### 14.4.14 enum flash\_swap\_control\_option\_t

Enumerator

*kFLASH\_SwapControlOptionIntializeSystem* Option used to Intialize Swap System.  
*kFLASH\_SwapControlOptionSetInUpdateState* Option used to Set Swap in Update State.  
*kFLASH\_SwapControlOptionSetInCompleteState* Option used to Set Swap in Complete State.  
*kFLASH\_SwapControlOptionReportStatus* Option used to Report Swap Status.  
*kFLASH\_SwapControlOptionDisableSystem* Option used to Disable Swap Status.

#### 14.4.15 enum flash\_swap\_state\_t

Enumerator

***kFLASH\_SwapStateUninitialized*** Flash swap system is in uninitialized state.

***kFLASH\_SwapStateReady*** Flash swap system is in ready state.

***kFLASH\_SwapStateUpdate*** Flash swap system is in update state.

***kFLASH\_SwapStateUpdateErased*** Flash swap system is in updateErased state.

***kFLASH\_SwapStateComplete*** Flash swap system is in complete state.

***kFLASH\_SwapStateDisabled*** Flash swap system is in disabled state.

#### 14.4.16 enum flash\_swap\_block\_status\_t

Enumerator

***kFLASH\_SwapBlockStatusLowerHalfProgramBlocksAtZero*** Swap block status is that lower half program block at zero.

***kFLASH\_SwapBlockStatusUpperHalfProgramBlocksAtZero*** Swap block status is that upper half program block at zero.

#### 14.4.17 enum flash\_partition\_flexram\_load\_option\_t

Enumerator

***kFLASH\_PartitionFlexramLoadOptionLoadedWithValidEepromData*** FlexRAM is loaded with valid EEPROM data during reset sequence.

***kFLASH\_PartitionFlexramLoadOptionNotLoaded*** FlexRAM is not loaded during reset sequence.

### 14.5 Function Documentation

#### 14.5.1 status\_t FLASH\_Init ( flash\_config\_t \* config )

This function checks and initializes Flash module for the other Flash APIs.

Parameters

<i>config</i>	Pointer to storage for the driver runtime state.
---------------	--

## Function Documentation

Return values

<i>kStatus_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FLASH_PartitionStatusUpdateFailure</i>	Failed to update partition status.

### 14.5.2 **status\_t FLASH\_SetCallback ( flash\_config\_t \* *config*, flash\_callback\_t *callback* )**

Parameters

<i>config</i>	Pointer to storage for the driver runtime state.
<i>callback</i>	callback function to be stored in driver

Return values

<i>kStatus_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.

### 14.5.3 **status\_t FLASH\_PreparesExecuteInRamFunctions ( flash\_config\_t \* *config* )**

Parameters

<i>config</i>	Pointer to storage for the driver runtime state.
---------------	--

Return values

<i>kStatus_FLASH_Success</i>	API was executed successfully.
------------------------------	--------------------------------

<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
--------------------------------------	-------------------------------

#### 14.5.4 status\_t FLASH\_EraseAll ( flash\_config\_t \* config, uint32\_t key )

Parameters

<i>config</i>	Pointer to storage for the driver runtime state.
<i>key</i>	value used to validate all flash erase APIs.

Return values

<i>kStatus_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FLASH_EraseKeyError</i>	API erase key is invalid.
<i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FLASH_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FLASH_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FLASH_CommandFailure</i>	Run-time error during command execution.
<i>kStatus_FLASH_PartitionStatusUpdateFailure</i>	Failed to update partition status

#### 14.5.5 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.

## Function Documentation

Parameters

<i>config</i>	Pointer to 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>	value used to validate all flash erase APIs.

Return values

<i>kStatus_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FLASH_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FLASH_AddressError</i>	Address is out of range.
<i>kStatus_FLASH_EraseKeyError</i>	API erase key is invalid.
<i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FLASH_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FLASH_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FLASH_CommandFailure</i>	Run-time error during command execution.

### 14.5.6 **status\_t FLASH\_EraseAllExecuteOnlySegments ( *flash\_config\_t \* config*, *uint32\_t key* )**

Parameters

<i>config</i>	Pointer to storage for the driver runtime state.
<i>key</i>	value used to validate all flash erase APIs.

Return values

<i>kStatus_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FLASH_EraseKeyError</i>	API erase key is invalid.
<i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FLASH_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FLASH_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FLASH_CommandFailure</i>	Run-time error during command execution.
<i>kStatus_FLASH_PartitionStatusUpdateFailure</i>	Failed to update partition status

Erases all program flash execute-only segments defined by the FXACC registers.

Parameters

<i>config</i>	Pointer to storage for the driver runtime state.
<i>key</i>	value used to validate all flash erase APIs.

Return values

<i>kStatus_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FLASH_EraseKeyError</i>	API erase key is invalid.
<i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.

## Function Documentation

<i>kStatus_FLASH_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FLASH_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FLASH_-CommandFailure</i>	Run-time error during command execution.

### 14.5.7 **status\_t FLASH\_Program ( flash\_config\_t \* config, uint32\_t start, uint32\_t \* src, uint32\_t lengthInBytes )**

This function programs the flash memory with desired data for a given flash area as determined by the start address and length.

#### Parameters

<i>config</i>	Pointer to 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>	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_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_Invalid-Argument</i>	Invalid argument is provided.
<i>kStatus_FLASH_-AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FLASH_Address-Error</i>	Address is out of range.
<i>kStatus_FLASH_Execute-InRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FLASH_Access-Error</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FLASH_-ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FLASH_-CommandFailure</i>	Run-time error during command execution.

#### 14.5.8 **status\_t FLASH\_ProgramOnce ( flash\_config\_t \* config, uint32\_t index, uint32\_t \* src, uint32\_t lengthInBytes )**

This function programs the Program Once Field with desired data for a given flash area as determined by the index and length.

Parameters

<i>config</i>	Pointer to storage for the driver runtime state.
<i>index</i>	The index indicating which area of Program Once Field to be programmed.
<i>src</i>	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_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FLASH_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FLASH_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FLASH_CommandFailure</i>	Run-time error during command execution.

#### 14.5.9 **status\_t FLASH\_ReadOnce ( flash\_config\_t \* config, uint32\_t index, uint32\_t \* dst, uint32\_t lengthInBytes )**

This function reads the flash memory with desired location for a given flash area as determined by the start address and length.

## Function Documentation

### Parameters

<i>config</i>	Pointer to 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>	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_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FLASH_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FLASH_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FLASH_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FLASH_CommandFailure</i>	Run-time error during command execution.

Read Program Once Field through parameters

This function reads the read once feild with given index and length

### Parameters

<i>config</i>	Pointer to storage for the driver runtime state.
<i>index</i>	The index indicating the area of program once field to be read.
<i>dst</i>	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_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FLASH_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FLASH_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FLASH_CommandFailure</i>	Run-time error during command execution.

#### 14.5.10 **status\_t FLASH\_GetSecurityState ( flash\_config\_t \* config, flash\_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>	Pointer to storage for the driver runtime state.
<i>state</i>	Pointer to the value returned for the current security status code:

Return values

<i>kStatus_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.

#### 14.5.11 **status\_t FLASH\_SecurityBypass ( flash\_config\_t \* config, const uint8\_t \* backdoorKey )**

If the MCU is in secured state, this function will unsecure the MCU by comparing the provided backdoor key with ones in the Flash Configuration Field.

## Function Documentation

Parameters

<i>config</i>	Pointer to storage for the driver runtime state.
<i>backdoorKey</i>	Pointer to the user buffer containing the backdoor key.

Return values

<i>kStatus_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FLASH_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FLASH_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FLASH_CommandFailure</i>	Run-time error during command execution.

### 14.5.12 **status\_t FLASH\_VerifyEraseAll ( flash\_config\_t \* config, flash\_margin\_value\_t margin )**

This function will check to see if the flash have been erased to the specified read margin level.

Parameters

<i>config</i>	Pointer to storage for the driver runtime state.
<i>margin</i>	Read margin choice

Return values

<i>kStatus_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FLASH_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FLASH_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FLASH_CommandFailure</i>	Run-time error during command execution.

#### 14.5.13 **status\_t FLASH\_VerifyErase ( flash\_config\_t \* config, uint32\_t start, uint32\_t lengthInBytes, flash\_margin\_value\_t margin )**

This function will check the appropriate number of flash sectors based on the desired start address and length to see if the flash have been erased to the specified read margin level.

Parameters

<i>config</i>	Pointer to 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_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.

## Function Documentation

<i>kStatus_FLASH_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FLASH_AddressError</i>	Address is out of range.
<i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FLASH_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FLASH_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FLASH_CommandFailure</i>	Run-time error during command execution.

**14.5.14 status\_t FLASH\_VerifyProgram ( flash\_config\_t \* config, uint32\_t start, uint32\_t lengthInBytes, const uint32\_t \* expectedData, flash\_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 with expected data for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	Pointer to 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>	Pointer to the expected data that is to be verified against.
<i>margin</i>	Read margin choice
<i>failedAddress</i>	Pointer to returned failing address.
<i>failedData</i>	Pointer to 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_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FLASH_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FLASH_AddressError</i>	Address is out of range.
<i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FLASH_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FLASH_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FLASH_CommandFailure</i>	Run-time error during command execution.

#### 14.5.15 **status\_t FLASH\_VerifyEraseAllExecuteOnlySegments ( flash\_config\_t \* config, flash\_margin\_value\_t margin )**

Parameters

<i>config</i>	Pointer to storage for the driver runtime state.
<i>margin</i>	Read margin choice

Return values

<i>kStatus_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.

## Function Documentation

<i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i>	Execute-in-RAM function is not available.
<i>kStatus_FLASH_AccessError</i>	Invalid instruction codes and out-of bounds addresses.
<i>kStatus_FLASH_ProtectionViolation</i>	The program/erase operation is requested to execute on protected areas.
<i>kStatus_FLASH_CommandFailure</i>	Run-time error during command execution.

### 14.5.16 **status\_t FLASH\_IsProtected ( flash\_config\_t \* config, uint32\_t start, uint32\_t lengthInBytes, flash\_protection\_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>	Pointer to 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>	Pointer to the value returned for the current protection status code for the desired flash area.

Return values

<i>kStatus_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FLASH_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FLASH_AddressError</i>	Address is out of range.

#### 14.5.17 **status\_t FLASH\_IsExecuteOnly ( flash\_config\_t \* config, uint32\_t start, uint32\_t lengthInBytes, flash\_execute\_only\_access\_state\_t \* access\_state )**

This function retrieves the current Flash access status for a given flash area as determined by the start address and length.

Parameters

<i>config</i>	Pointer to storage for the driver runtime state.
<i>start</i>	The start address of the desired flash memory to be checked. Must be word-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words) to be checked. Must be word-aligned.
<i>access_state</i>	Pointer to the value returned for the current access status code for the desired flash area.

Return values

<i>kStatus_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FLASH_AlignmentError</i>	Parameter is not aligned with specified baseline.
<i>kStatus_FLASH_AddressError</i>	Address is out of range.

#### 14.5.18 **status\_t FLASH\_GetProperty ( flash\_config\_t \* config, flash\_property\_tag\_t whichProperty, uint32\_t \* value )**

Parameters

<i>config</i>	Pointer to storage for the driver runtime state.
<i>whichProperty</i>	The desired property from the list of properties in enum flash_property_tag_t

## Function Documentation

<i>value</i>	Pointer to the value returned for the desired flash property
--------------	--

Return values

<i>kStatus_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FLASH_-UnknownProperty</i>	unknown property tag

### 14.5.19 **status\_t FLASH\_PflashSetProtection ( flash\_config\_t \* *config*, uint32\_t *protectStatus* )**

Parameters

<i>config</i>	Pointer to storage for the driver runtime state.
<i>protectStatus</i>	The expected protect status user wants to set to PFlash protection register. Each bit is corresponding to protection of 1/32 of the total PFlash. The least significant bit is corresponding to the lowest address area of P-Flash. 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_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_InvalidArgument</i>	Invalid argument is provided.
<i>kStatus_FLASH_-CommandFailure</i>	Run-time error during command execution.

### 14.5.20 status\_t FLASH\_PflashGetProtection ( flash\_config\_t \* *config*, uint32\_t \* *protectStatus* )

Parameters

<i>config</i>	Pointer to storage for the driver runtime state.
<i>protectStatus</i>	Protect status returned by PFlash IP. Each bit is corresponding to protection of 1/32 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 below: 0: this area is protected. 1: this area is unprotected.

Return values

<i>kStatus_FLASH_Success</i>	API was executed successfully.
<i>kStatus_FLASH_Invalid-Argument</i>	Invalid argument is provided.

## Function Documentation

# Chapter 15

## FlexIO: FlexIO Driver

### 15.1 Overview

The KSDK provides a generic driver for the FlexIO module of Kinetis devices, as well as multiple protocol-specific FlexIO drivers.

### Modules

- [FlexIO Camera Driver](#)
- [FlexIO Driver](#)
- [FlexIO I2C Master Driver](#)
- [FlexIO I2S Driver](#)
- [FlexIO SPI Driver](#)
- [FlexIO UART Driver](#)

## FlexIO Driver

### 15.2 FlexIO Driver

#### 15.2.1 Overview

## Data Structures

- struct `flexio_config_t`  
*Define FlexIO user configuration structure. [More...](#)*
- struct `flexio_timer_config_t`  
*Define FlexIO timer configuration structure. [More...](#)*
- struct `flexio_shifter_config_t`  
*Define FlexIO shifter configuration structure. [More...](#)*

## Macros

- #define `FLEXIO_TIMER_TRIGGER_SEL_PININPUT`(x) ((uint32\_t)(x) << 1U)  
*Calculate FlexIO timer trigger.*

## Typedefs

- typedef void(\* `flexio_isr_t` )(void \*base, void \*handle)  
*typedef for FlexIO simulated driver interrupt handler.*

## Enumerations

- enum `flexio_timer_trigger_polarity_t` {  
  `kFLEXIO_TimerTriggerPolarityActiveHigh` = 0x0U,  
  `kFLEXIO_TimerTriggerPolarityActiveLow` = 0x1U }  
*Define time of timer trigger polarity.*
- enum `flexio_timer_trigger_source_t` {  
  `kFLEXIO_TimerTriggerSourceExternal` = 0x0U,  
  `kFLEXIO_TimerTriggerSourceInternal` = 0x1U }  
*Define type of timer trigger source.*
- enum `flexio_pin_config_t` {  
  `kFLEXIO_PinConfigOutputDisabled` = 0x0U,  
  `kFLEXIO_PinConfigOpenDrainOrBidirection` = 0x1U,  
  `kFLEXIO_PinConfigBidirectionOutputData` = 0x2U,  
  `kFLEXIO_PinConfigOutput` = 0x3U }  
*Define type of timer/shifter pin configuration.*
- enum `flexio_pin_polarity_t` {  
  `kFLEXIO_PinActiveHigh` = 0x0U,  
  `kFLEXIO_PinActiveLow` = 0x1U }  
*Definition of pin polarity.*

- enum `flexio_timer_mode_t` {
   
  `kFLEXIO_TimerModeDisabled` = 0x0U,
   
  `kFLEXIO_TimerModeDual8BitBaudBit` = 0x1U,
   
  `kFLEXIO_TimerModeDual8BitPWM` = 0x2U,
   
  `kFLEXIO_TimerModeSingle16Bit` = 0x3U }

*Define type of timer work mode.*

- enum `flexio_timer_output_t` {
   
  `kFLEXIO_TimerOutputOneNotAffectedByReset` = 0x0U,
   
  `kFLEXIO_TimerOutputZeroNotAffectedByReset` = 0x1U,
   
  `kFLEXIO_TimerOutputOneAffectedByReset` = 0x2U,
   
  `kFLEXIO_TimerOutputZeroAffectedByReset` = 0x3U }

*Define type of timer initial output or timer reset condition.*

- enum `flexio_timer_decrement_source_t` {
   
  `kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput` = 0x0U,
   
  `kFLEXIO_TimerDecSrcOnTriggerInputShiftTimerOutput` = 0x1U,
   
  `kFLEXIO_TimerDecSrcOnPinInputShiftPinInput` = 0x2U,
   
  `kFLEXIO_TimerDecSrcOnTriggerInputShiftTriggerInput` = 0x3U }

*Define type of timer decrement.*

- enum `flexio_timer_reset_condition_t` {
   
  `kFLEXIO_TimerResetNever` = 0x0U,
   
  `kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput` = 0x2U,
   
  `kFLEXIO_TimerResetOnTimerTriggerEqualToTimerOutput` = 0x3U,
   
  `kFLEXIO_TimerResetOnTimerPinRisingEdge` = 0x4U,
   
  `kFLEXIO_TimerResetOnTimerTriggerRisingEdge` = 0x6U,
   
  `kFLEXIO_TimerResetOnTimerTriggerBothEdge` = 0x7U }

*Define type of timer reset condition.*

- enum `flexio_timer_disable_condition_t` {
   
  `kFLEXIO_TimerDisableNever` = 0x0U,
   
  `kFLEXIO_TimerDisableOnPreTimerDisable` = 0x1U,
   
  `kFLEXIO_TimerDisableOnTimerCompare` = 0x2U,
   
  `kFLEXIO_TimerDisableOnTimerCompareTriggerLow` = 0x3U,
   
  `kFLEXIO_TimerDisableOnPinBothEdge` = 0x4U,
   
  `kFLEXIO_TimerDisableOnPinBothEdgeTriggerHigh` = 0x5U,
   
  `kFLEXIO_TimerDisableOnTriggerFallingEdge` = 0x6U }

*Define type of timer disable condition.*

- enum `flexio_timer_enable_condition_t` {
   
  `kFLEXIO_TimerEnabledAlways` = 0x0U,
   
  `kFLEXIO_TimerEnableOnPrevTimerEnable` = 0x1U,
   
  `kFLEXIO_TimerEnableOnTriggerHigh` = 0x2U,
   
  `kFLEXIO_TimerEnableOnTriggerHighPinHigh` = 0x3U,
   
  `kFLEXIO_TimerEnableOnPinRisingEdge` = 0x4U,
   
  `kFLEXIO_TimerEnableOnPinRisingEdgeTriggerHigh` = 0x5U,
   
  `kFLEXIO_TimerEnableOnTriggerRisingEdge` = 0x6U,
   
  `kFLEXIO_TimerEnableOnTriggerBothEdge` = 0x7U }

*Define type of timer enable condition.*

- enum `flexio_timer_stop_bit_condition_t` {

## FlexIO Driver

```
kFLEXIO_TimerStopBitDisabled = 0x0U,  
kFLEXIO_TimerStopBitEnableOnTimerCompare = 0x1U,  
kFLEXIO_TimerStopBitEnableOnTimerDisable = 0x2U,  
kFLEXIO_TimerStopBitEnableOnTimerCompareDisable = 0x3U }
```

*Define type of timer stop bit generate condition.*

- enum `flexio_timer_start_bit_condition_t`{  
    kFLEXIO\_TimerStartBitDisabled = 0x0U,  
    kFLEXIO\_TimerStartBitEnabled = 0x1U }

*Define type of timer start bit generate condition.*

- enum `flexio_shifter_timer_polarity_t`  
*Define type of timer polarity for shifter control.*
- enum `flexio_shifter_mode_t`{

```
    kFLEXIO_ShifterDisabled = 0x0U,  
    kFLEXIO_ShifterModeReceive = 0x1U,  
    kFLEXIO_ShifterModeTransmit = 0x2U,  
    kFLEXIO_ShifterModeMatchStore = 0x4U,  
    kFLEXIO_ShifterModeMatchContinuous = 0x5U }
```

*Define type of shifter working mode.*

- enum `flexio_shifter_input_source_t`{  
    kFLEXIO\_ShifterInputFromPin = 0x0U,  
    kFLEXIO\_ShifterInputFromNextShifterOutput = 0x1U }

*Define type of shifter input source.*

- enum `flexio_shifter_stop_bit_t`{  
    kFLEXIO\_ShifterStopBitDisable = 0x0U,  
    kFLEXIO\_ShifterStopBitLow = 0x2U,  
    kFLEXIO\_ShifterStopBitHigh = 0x3U }

*Define of STOP bit configuration.*

- enum `flexio_shifter_start_bit_t`{  
    kFLEXIO\_ShifterStartBitDisabledLoadDataOnEnable = 0x0U,  
    kFLEXIO\_ShifterStartBitDisabledLoadDataOnShift = 0x1U,  
    kFLEXIO\_ShifterStartBitLow = 0x2U,  
    kFLEXIO\_ShifterStartBitHigh = 0x3U }

*Define type of START bit configuration.*

- enum `flexio_shifter_buffer_type_t`{  
    kFLEXIO\_ShifterBuffer = 0x0U,  
    kFLEXIO\_ShifterBufferBitSwapped = 0x1U,  
    kFLEXIO\_ShifterBufferByteSwapped = 0x2U,  
    kFLEXIO\_ShifterBufferBitByteSwapped = 0x3U }

*Define FlexIO shifter buffer type.*

## Driver version

- #define `FSL_FLEXIO_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 0)`)  
*FlexIO driver version 2.0.0.*

## FlexIO Initialization and De-initialization

- void [FLEXIO\\_GetDefaultConfig](#) ([flexio\\_config\\_t](#) \*userConfig)  
*Gets the default configuration to configure FlexIO module.*
- void [FLEXIO\\_Init](#) ([FLEXIO\\_Type](#) \*base, const [flexio\\_config\\_t](#) \*userConfig)  
*Configures the FlexIO with FlexIO configuration.*
- void [FLEXIO\\_Deinit](#) ([FLEXIO\\_Type](#) \*base)  
*Gates the FlexIO clock.*

## FlexIO Basic Operation

- void [FLEXIO\\_Reset](#) ([FLEXIO\\_Type](#) \*base)  
*Resets the FlexIO module.*
- static void [FLEXIO\\_Enable](#) ([FLEXIO\\_Type](#) \*base, bool enable)  
*Enables the FlexIO module operation.*
- void [FLEXIO\\_SetShifterConfig](#) ([FLEXIO\\_Type](#) \*base, uint8\_t index, const [flexio\\_shifter\\_config\\_t](#) \*shifterConfig)  
*Configures the shifter with shifter configuration.*
- void [FLEXIO\\_SetTimerConfig](#) ([FLEXIO\\_Type](#) \*base, uint8\_t index, const [flexio\\_timer\\_config\\_t](#) \*timerConfig)  
*Configures the timer with the timer configuration.*

## FlexIO Interrupt Operation

- static void [FLEXIO\\_EnableShifterStatusInterrupts](#) ([FLEXIO\\_Type](#) \*base, uint32\_t mask)  
*Enables the shifter status interrupt.*
- static void [FLEXIO\\_DisableShifterStatusInterrupts](#) ([FLEXIO\\_Type](#) \*base, uint32\_t mask)  
*Disables the shifter status interrupt.*
- static void [FLEXIO\\_EnableShifterErrorInterrupts](#) ([FLEXIO\\_Type](#) \*base, uint32\_t mask)  
*Enables the shifter error interrupt.*
- static void [FLEXIO\\_DisableShifterErrorInterrupts](#) ([FLEXIO\\_Type](#) \*base, uint32\_t mask)  
*Disables the shifter error interrupt.*
- static void [FLEXIO\\_EnableTimerStatusInterrupts](#) ([FLEXIO\\_Type](#) \*base, uint32\_t mask)  
*Enables the timer status interrupt.*
- static void [FLEXIO\\_DisableTimerStatusInterrupts](#) ([FLEXIO\\_Type](#) \*base, uint32\_t mask)  
*Disables the timer status interrupt.*

## FlexIO Status Operation

- static uint32\_t [FLEXIO\\_GetShifterStatusFlags](#) ([FLEXIO\\_Type](#) \*base)  
*Gets the shifter status flags.*
- static void [FLEXIO\\_ClearShifterStatusFlags](#) ([FLEXIO\\_Type](#) \*base, uint32\_t mask)  
*Clears the shifter status flags.*
- static uint32\_t [FLEXIO\\_GetShifterErrorFlags](#) ([FLEXIO\\_Type](#) \*base)  
*Gets the shifter error flags.*
- static void [FLEXIO\\_ClearShifterErrorFlags](#) ([FLEXIO\\_Type](#) \*base, uint32\_t mask)  
*Clears the shifter error flags.*

## FlexIO Driver

- static uint32\_t **FLEXIO\_GetTimerStatusFlags** (FLEXIO\_Type \*base)  
*Gets the timer status flags.*
- static void **FLEXIO\_ClearTimerStatusFlags** (FLEXIO\_Type \*base, uint32\_t mask)  
*Clears the timer status flags.*

## FlexIO DMA Operation

- static void **FLEXIO\_EnableShifterStatusDMA** (FLEXIO\_Type \*base, uint32\_t mask, bool enable)  
*Enables/disables the shifter status DMA.*
- uint32\_t **FLEXIO\_GetShifterBufferAddress** (FLEXIO\_Type \*base, flexio\_shifter\_buffer\_type\_t type, uint8\_t index)  
*Gets the shifter buffer address for the DMA transfer usage.*
- status\_t **FLEXIO\_RegisterHandleIRQ** (void \*base, void \*handle, flexio\_isr\_t isr)  
*Registers the handle and the interrupt handler for the FlexIO-simulated peripheral.*
- status\_t **FLEXIO\_UnregisterHandleIRQ** (void \*base)  
*Unregisters the handle and the interrupt handler for the FlexIO-simulated peripheral.*

### 15.2.2 Data Structure Documentation

#### 15.2.2.1 struct flexio\_config\_t

##### Data Fields

- bool **enableFlexio**  
*Enable/disable FlexIO module.*
- bool **enableInDoze**  
*Enable/disable FlexIO operation in doze mode.*
- bool **enableInDebug**  
*Enable/disable FlexIO operation in debug mode.*
- bool **enableFastAccess**  
*Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*

##### 15.2.2.1.0.38 Field Documentation

###### 15.2.2.1.0.38.1 bool flexio\_config\_t::enableFastAccess

#### 15.2.2.2 struct flexio\_timer\_config\_t

##### Data Fields

- uint32\_t **triggerSelect**  
*The internal trigger selection number using MACROs.*
- flexio\_timer\_trigger\_polarity\_t **triggerPolarity**  
*Trigger Polarity.*
- flexio\_timer\_trigger\_source\_t **triggerSource**  
*Trigger Source, internal (see 'trgsel') or external.*
- flexio\_pin\_config\_t **pinConfig**

*Timer Pin Configuration.*

- **uint32\_t pinSelect**  
*Timer Pin number Select.*
- **flexio\_pin\_polarity\_t pinPolarity**  
*Timer Pin Polarity.*
- **flexio\_timer\_mode\_t timerMode**  
*Timer work Mode.*
- **flexio\_timer\_output\_t timerOutput**  
Configures the initial state of the Timer Output and  
*whether it is affected by the Timer reset.*
- **flexio\_timer\_decrement\_source\_t timerDecrement**  
Configures the source of the Timer decrement and the  
*source of the Shift clock.*
- **flexio\_timer\_reset\_condition\_t timerReset**  
Configures the condition that causes the timer counter  
*(and optionally the timer output) to be reset.*
- **flexio\_timer\_disable\_condition\_t timerDisable**  
Configures the condition that causes the Timer to be  
*disabled and stop decrementing.*
- **flexio\_timer\_enable\_condition\_t timerEnable**  
Configures the condition that causes the Timer to be  
*enabled and start decrementing.*
- **flexio\_timer\_stop\_bit\_condition\_t timerStop**  
*Timer STOP Bit generation.*
- **flexio\_timer\_start\_bit\_condition\_t timerStart**  
*Timer STRAT Bit generation.*
- **uint32\_t timerCompare**  
*Value for Timer Compare N Register.*

## FlexIO Driver

### 15.2.2.2.0.39 Field Documentation

15.2.2.2.0.39.1 `uint32_t flexio_timer_config_t::triggerSelect`

15.2.2.2.0.39.2 `flexio_timer_polarity_t flexio_timer_config_t::triggerPolarity`

15.2.2.2.0.39.3 `flexio_timer_trigger_source_t flexio_timer_config_t::triggerSource`

15.2.2.2.0.39.4 `flexio_pin_config_t flexio_timer_config_t::pinConfig`

15.2.2.2.0.39.5 `uint32_t flexio_timer_config_t::pinSelect`

15.2.2.2.0.39.6 `flexio_pin_polarity_t flexio_timer_config_t::pinPolarity`

15.2.2.2.0.39.7 `flexio_timer_mode_t flexio_timer_config_t::timerMode`

15.2.2.2.0.39.8 `flexio_timer_output_t flexio_timer_config_t::timerOutput`

15.2.2.2.0.39.9 `flexio_timer_decrement_source_t flexio_timer_config_t::timerDecrement`

15.2.2.2.0.39.10 `flexio_timer_reset_condition_t flexio_timer_config_t::timerReset`

15.2.2.2.0.39.11 `flexio_timer_disable_condition_t flexio_timer_config_t::timerDisable`

15.2.2.2.0.39.12 `flexio_timer_enable_condition_t flexio_timer_config_t::timerEnable`

15.2.2.2.0.39.13 `flexio_timer_stop_bit_condition_t flexio_timer_config_t::timerStop`

15.2.2.2.0.39.14 `flexio_timer_start_bit_condition_t flexio_timer_config_t::timerStart`

15.2.2.2.0.39.15 `uint32_t flexio_timer_config_t::timerCompare`

### 15.2.2.3 `struct flexio_shifter_config_t`

#### Data Fields

- `uint32_t timerSelect`  
*Selects which Timer is used for controlling the logic/shift register and generating the Shift clock.*
- `flexio_shifter_timer_polarity_t timerPolarity`  
*Timer Polarity.*
- `flexio_pin_config_t pinConfig`  
*Shifter Pin Configuration.*
- `uint32_t pinSelect`  
*Shifter Pin number Select.*
- `flexio_pin_polarity_t pinPolarity`  
*Shifter Pin Polarity.*
- `flexio_shifter_mode_t shifterMode`  
*Configures the mode of the Shifter.*
- `flexio_shifter_input_source_t inputSource`  
*Selects the input source for the shifter.*

- `flexio_shifter_stop_bit_t shifterStop`  
*Shifter STOP bit.*
- `flexio_shifter_start_bit_t shifterStart`  
*Shifter START bit.*

#### 15.2.2.3.0.40 Field Documentation

15.2.2.3.0.40.1 `uint32_t flexio_shifter_config_t::timerSelect`

15.2.2.3.0.40.2 `flexio_shifter_timer_polarity_t flexio_shifter_config_t::timerPolarity`

15.2.2.3.0.40.3 `flexio_pin_config_t flexio_shifter_config_t::pinConfig`

15.2.2.3.0.40.4 `uint32_t flexio_shifter_config_t::pinSelect`

15.2.2.3.0.40.5 `flexio_pin_polarity_t flexio_shifter_config_t::pinPolarity`

15.2.2.3.0.40.6 `flexio_shifter_mode_t flexio_shifter_config_t::shifterMode`

15.2.2.3.0.40.7 `flexio_shifter_input_source_t flexio_shifter_config_t::inputSource`

15.2.2.3.0.40.8 `flexio_shifter_stop_bit_t flexio_shifter_config_t::shifterStop`

15.2.2.3.0.40.9 `flexio_shifter_start_bit_t flexio_shifter_config_t::shifterStart`

#### 15.2.3 Macro Definition Documentation

15.2.3.1 `#define FSL_FLEXIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

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

#### 15.2.4 Typedef Documentation

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

#### 15.2.5 Enumeration Type Documentation

15.2.5.1 `enum flexio_timer_trigger_polarity_t`

Enumerator

`kFLEXIO_TimerTriggerPolarityActiveHigh` Active high.

`kFLEXIO_TimerTriggerPolarityActiveLow` Active low.

## FlexIO Driver

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

Enumerator

*kFLEXIO\_TimerTriggerSourceExternal* External trigger selected.

*kFLEXIO\_TimerTriggerSourceInternal* Internal trigger selected.

### 15.2.5.3 enum flexio\_pin\_config\_t

Enumerator

*kFLEXIO\_PinConfigOutputDisabled* Pin output disabled.

*kFLEXIO\_PinConfigOpenDrainOrBidirection* Pin open drain or bidirectional output enable.

*kFLEXIO\_PinConfigBidirectionOutputData* Pin bidirectional output data.

*kFLEXIO\_PinConfigOutput* Pin output.

### 15.2.5.4 enum flexio\_pin\_polarity\_t

Enumerator

*kFLEXIO\_PinActiveHigh* Active high.

*kFLEXIO\_PinActiveLow* Active low.

### 15.2.5.5 enum flexio\_timer\_mode\_t

Enumerator

*kFLEXIO\_TimerModeDisabled* Timer Disabled.

*kFLEXIO\_TimerModeDual8BitBaudBit* Dual 8-bit counters baud/bit mode.

*kFLEXIO\_TimerModeDual8BitPWM* Dual 8-bit counters PWM mode.

*kFLEXIO\_TimerModeSingle16Bit* Single 16-bit counter mode.

### 15.2.5.6 enum flexio\_timer\_output\_t

Enumerator

*kFLEXIO\_TimerOutputOneNotAffectedByReset* Logic one when enabled and is not affected by timer reset.

*kFLEXIO\_TimerOutputZeroNotAffectedByReset* Logic zero when enabled and is not affected by timer reset.

*kFLEXIO\_TimerOutputOneAffectedByReset* Logic one when enabled and on timer reset.

*kFLEXIO\_TimerOutputZeroAffectedByReset* Logic zero when enabled and on timer reset.

### 15.2.5.7 enum flexio\_timer\_decrement\_source\_t

Enumerator

- kFLEXIO\_TimerDecSrcOnFlexIOClockShiftTimerOutput*** Decrement counter on FlexIO clock, Shift clock equals Timer output.
- kFLEXIO\_TimerDecSrcOnTriggerInputShiftTimerOutput*** Decrement counter on Trigger input (both edges), Shift clock equals Timer output.
- kFLEXIO\_TimerDecSrcOnPinInputShiftPinInput*** Decrement counter on Pin input (both edges), Shift clock equals Pin input.
- kFLEXIO\_TimerDecSrcOnTriggerInputShiftTriggerInput*** Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

### 15.2.5.8 enum flexio\_timer\_reset\_condition\_t

Enumerator

- kFLEXIO\_TimerResetNever*** Timer never reset.
- kFLEXIO\_TimerResetOnTimerPinEqualToTimerOutput*** Timer reset on Timer Pin equal to Timer Output.
- kFLEXIO\_TimerResetOnTimerTriggerEqualToTimerOutput*** Timer reset on Timer Trigger equal to Timer Output.
- kFLEXIO\_TimerResetOnTimerPinRisingEdge*** Timer reset on Timer Pin rising edge.
- kFLEXIO\_TimerResetOnTimerTriggerRisingEdge*** Timer reset on Trigger rising edge.
- kFLEXIO\_TimerResetOnTimerTriggerBothEdge*** Timer reset on Trigger rising or falling edge.

### 15.2.5.9 enum flexio\_timer\_disable\_condition\_t

Enumerator

- kFLEXIO\_TimerDisableNever*** Timer never disabled.
- kFLEXIO\_TimerDisableOnPreTimerDisable*** Timer disabled on Timer N-1 disable.
- kFLEXIO\_TimerDisableOnTimerCompare*** Timer disabled on Timer compare.
- kFLEXIO\_TimerDisableOnTimerCompareTriggerLow*** Timer disabled on Timer compare and Trigger Low.
- kFLEXIO\_TimerDisableOnPinBothEdge*** Timer disabled on Pin rising or falling edge.
- kFLEXIO\_TimerDisableOnPinBothEdgeTriggerHigh*** Timer disabled on Pin rising or falling edge provided Trigger is high.
- kFLEXIO\_TimerDisableOnTriggerFallingEdge*** Timer disabled on Trigger falling edge.

### 15.2.5.10 enum flexio\_timer\_enable\_condition\_t

Enumerator

- kFLEXIO\_TimerEnabledAlways*** Timer always enabled.

## FlexIO Driver

***kFLEXIO\_TimerEnableOnPrevTimerEnable*** Timer enabled on Timer N-1 enable.  
***kFLEXIO\_TimerEnableOnTriggerHigh*** Timer enabled on Trigger high.  
***kFLEXIO\_TimerEnableOnTriggerHighPinHigh*** Timer enabled on Trigger high and Pin high.  
***kFLEXIO\_TimerEnableOnPinRisingEdge*** Timer enabled on Pin rising edge.  
***kFLEXIO\_TimerEnableOnPinRisingEdgeTriggerHigh*** Timer enabled on Pin rising edge and Trigger high.  
***kFLEXIO\_TimerEnableOnTriggerRisingEdge*** Timer enabled on Trigger rising edge.  
***kFLEXIO\_TimerEnableOnTriggerBothEdge*** Timer enabled on Trigger rising or falling edge.

### 15.2.5.11 enum flexio\_timer\_stop\_bit\_condition\_t

Enumerator

***kFLEXIO\_TimerStopBitDisabled*** Stop bit disabled.  
***kFLEXIO\_TimerStopBitEnableOnTimerCompare*** Stop bit is enabled on timer compare.  
***kFLEXIO\_TimerStopBitEnableOnTimerDisable*** Stop bit is enabled on timer disable.  
***kFLEXIO\_TimerStopBitEnableOnTimerCompareDisable*** Stop bit is enabled on timer compare and timer disable.

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

Enumerator

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

### 15.2.5.13 enum flexio\_shifter\_timer\_polarity\_t

### 15.2.5.14 enum flexio\_shifter\_mode\_t

Enumerator

***kFLEXIO\_ShifterDisabled*** Shifter is disabled.  
***kFLEXIO\_ShifterModeReceive*** Receive mode.  
***kFLEXIO\_ShifterModeTransmit*** Transmit mode.  
***kFLEXIO\_ShifterModeMatchStore*** Match store mode.  
***kFLEXIO\_ShifterModeMatchContinuous*** Match continuous mode.

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

Enumerator

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

### 15.2.5.16 enum flexio\_shifter\_stop\_bit\_t

Enumerator

***kFLEXIO\_ShifterStopBitDisable*** Disable shifter stop bit.

***kFLEXIO\_ShifterStopBitLow*** Set shifter stop bit to logic low level.

***kFLEXIO\_ShifterStopBitHigh*** Set shifter stop bit to logic high level.

### 15.2.5.17 enum flexio\_shifter\_start\_bit\_t

Enumerator

***kFLEXIO\_ShifterStartBitDisabledLoadDataOnEnable*** Disable shifter start bit, transmitter loads data on enable.

***kFLEXIO\_ShifterStartBitDisabledLoadDataOnShift*** Disable shifter start bit, transmitter loads data on first shift.

***kFLEXIO\_ShifterStartBitLow*** Set shifter start bit to logic low level.

***kFLEXIO\_ShifterStartBitHigh*** Set shifter start bit to logic high level.

### 15.2.5.18 enum flexio\_shifter\_buffer\_type\_t

Enumerator

***kFLEXIO\_ShifterBuffer*** Shifter Buffer N Register.

***kFLEXIO\_ShifterBufferBitSwapped*** Shifter Buffer N Bit Byte Swapped Register.

***kFLEXIO\_ShifterBufferByteSwapped*** Shifter Buffer N Byte Swapped Register.

***kFLEXIO\_ShifterBufferBitByteSwapped*** Shifter Buffer N Bit Swapped Register.

## 15.2.6 Function Documentation

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

The configuration can used directly for calling FLEXIO\_Configure().

Example:

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

## FlexIO Driver

Parameters

<i>userConfig</i>	pointer to <a href="#">flexio_config_t</a> structure
-------------------	--

### 15.2.6.2 void FLEXIO\_Init ( [FLEXIO\\_Type](#) \* *base*, const [flexio\\_config\\_t](#) \* *userConfig* )

The configuration structure can be filled by the user, or be set with default values by [FLEXIO\\_GetDefaultConfig\(\)](#).

Example

```
flexio_config_t config = {  
    .enableFlexio = true,  
    .enableInDoze = false,  
    .enableInDebug = true,  
    .enableFastAccess = false  
};  
FLEXIO_Configure(base, &config);
```

Parameters

<i>base</i>	FlexIO peripheral base address
<i>userConfig</i>	pointer to <a href="#">flexio_config_t</a> structure

### 15.2.6.3 void FLEXIO\_Deinit ( [FLEXIO\\_Type](#) \* *base* )

Call this API to stop the FlexIO clock.

Note

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

Parameters

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

### 15.2.6.4 void FLEXIO\_Reset ( [FLEXIO\\_Type](#) \* *base* )

Parameters

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

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

Parameters

<i>base</i>	FlexIO peripheral base address
<i>enable</i>	true to enable, false to disable.

### 15.2.6.6 void FLEXIO\_SetShifterConfig ( FLEXIO\_Type \* *base*, uint8\_t *index*, const flexio\_shifter\_config\_t \* *shifterConfig* )

The configuration structure covers both the SHIFTCTL and SHIFTCFG registers. To configure the shifter to the proper mode, select which timer controls the shifter to shift, whether to generate start bit/stop bit, and the polarity of start bit and stop bit.

Example

```
flexio_shifter_config_t config = {
.timerSelect = 0,
.timerPolarity = kFLEXIO_ShifterTimerPolarityOnPositive,
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
.pinPolarity = kFLEXIO_PinActiveLow,
.shifterMode = kFLEXIO_ShifterModeTransmit,
.inputSource = kFLEXIO_ShifterInputFromPin,
.shifterStop = kFLEXIO_ShifterStopBitHigh,
.shifterStart = kFLEXIO_ShifterStartBitLow
};
FLEXIO_SetShifterConfig(base, &config);
```

Parameters

<i>base</i>	FlexIO peripheral base address
<i>index</i>	shifter index
<i>shifterConfig</i>	pointer to <a href="#">flexio_shifter_config_t</a> structure

### 15.2.6.7 void FLEXIO\_SetTimerConfig ( FLEXIO\_Type \* *base*, uint8\_t *index*, const flexio\_timer\_config\_t \* *timerConfig* )

The configuration structure covers both the TIMCTL and TIMCFG registers. To configure the timer to the proper mode, select trigger source for timer and the timer pin output and the timing for timer.

## FlexIO Driver

### Example

```
flexio_timer_config_t config = {  
    .triggerSelect = FLEXIO_TIMER_TRIGGER_SEL_SHIFTnSTAT(0),  
    .triggerPolarity = kFLEXIO_TimerTriggerPolarityActiveLow,  
    .triggerSource = kFLEXIO_TimerTriggerSourceInternal,  
    .pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,  
    .pinSelect = 0,  
    .pinPolarity = kFLEXIO_PinActiveHigh,  
    .timerMode = kFLEXIO_TimerModeDual8BitBaudBit,  
    .timerOutput = kFLEXIO_TimerOutputZeroNotAffectedByReset,  
    .timerDecrement = kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput  
    ,  
    .timerReset = kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput,  
    .timerDisable = kFLEXIO_TimerDisableOnTimerCompare,  
    .timerEnable = kFLEXIO_TimerEnableOnTriggerHigh,  
    .timerStop = kFLEXIO_TimerStopBitEnableOnTimerDisable,  
    .timerStart = kFLEXIO_TimerStartBitEnabled  
};  
FLEXIO_SetTimerConfig(base, &config);
```

### Parameters

<i>base</i>	FlexIO peripheral base address
<i>index</i>	timer index
<i>timerConfig</i>	pointer to <a href="#">flexio_timer_config_t</a> structure

### 15.2.6.8 static void FLEXIO\_EnableShifterStatusInterrupts ( FLEXIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

The interrupt generates when the corresponding SSF is set.

### Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	the shifter status mask which could be calculated by ( $1 \ll$ shifter index)

### Note

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

### 15.2.6.9 static void FLEXIO\_DisableShifterStatusInterrupts ( FLEXIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

The interrupt won't generate when the corresponding SSF is set.

Parameters

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

Note

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

#### 15.2.6.10 static void FLEXIO\_EnableShifterErrorInterrupts ( FLEXIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

The interrupt generates when the corresponding SEF is set.

Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	the shifter error mask which could be calculated by ( $1 << \text{shifter index}$ )

Note

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

#### 15.2.6.11 static void FLEXIO\_DisableShifterErrorInterrupts ( FLEXIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

The interrupt won't generate when the corresponding SEF is set.

Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	the shifter error mask which could be calculated by ( $1 << \text{shifter index}$ )

Note

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

**15.2.6.12 static void FLEXIO\_EnableTimerStatusInterrupts ( FLEXIO\_Type \* *base*,  
                  uint32\_t *mask* ) [inline], [static]**

The interrupt generates when the corresponding SSF is set.

Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	the timer status mask which could be calculated by ( $1 << \text{timer index}$ )

Note

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

#### 15.2.6.13 static void FLEXIO\_DisableTimerStatusInterrupts ( FLEXIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

The interrupt won't generate when the corresponding SSF is set.

Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	the timer status mask which could be calculated by ( $1 << \text{timer index}$ )

Note

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

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

Parameters

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

Returns

shifter status flags

#### 15.2.6.15 static void FLEXIO\_ClearShifterStatusFlags ( FLEXIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## FlexIO Driver

Parameters

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

Note

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

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

Parameters

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

Returns

shifter error flags

### 15.2.6.17 static void FLEXIO\_ClearShifterErrorFlags ( FLEXIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	the shifter error mask which could be calculated by ( $1 << \text{shifter index}$ )

Note

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

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

Parameters

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

Returns

timer status flags

#### 15.2.6.19 static void FLEXIO\_ClearTimerStatusFlags ( **FLEXIO\_Type** \* *base*, **uint32\_t** *mask* ) [inline], [static]

Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	the timer status mask which could be calculated by (1 << timer index)

Note

for clearing multiple timer status flags, for example, two timer status flags, could calculate the mask by using ((1 << timer index0) | (1 << timer index1))

#### 15.2.6.20 static void FLEXIO\_EnableShifterStatusDMA ( **FLEXIO\_Type** \* *base*, **uint32\_t** *mask*, **bool enable** ) [inline], [static]

The DMA request generates when the corresponding SSF is set.

Note

For multiple shifter status DMA enables, for example, calculate the mask by using ((1 << shifter index0) | (1 << shifter index1))

Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	the shifter status mask which could be calculated by (1 << shifter index)

## FlexIO Driver

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

**15.2.6.21 `uint32_t FLEXIO_GetShifterBufferAddress ( FLEXIO_Type * base, flexio_shifter_buffer_type_t type, uint8_t index )`**

Parameters

<i>base</i>	FlexIO peripheral base address
<i>type</i>	shifter type of flexio_shifter_buffer_type_t
<i>index</i>	shifter index

Returns

corresponding shifter buffer index

**15.2.6.22 `status_t FLEXIO_RegisterHandleIRQ ( void * base, void * handle, flexio_isr_t isr )`**

Parameters

<i>base</i>	pointer to FlexIO simulated peripheral type.
<i>handle</i>	pointer to handler for FlexIO simulated peripheral.
<i>isr</i>	FlexIO simulated peripheral interrupt handler.

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_OutOfRange</i>	The FlexIO type/handle/ISR table out of range.

**15.2.6.23 `status_t FLEXIO_UnregisterHandleIRQ ( void * base )`**

Parameters

<i>base</i>	pointer to FlexIO simulated peripheral type.
-------------	--

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_OutOfRange</i>	The FlexIO type/handle/ISR table out of range.

### 15.3 FlexIO Camera Driver

#### 15.3.1 Overview

The KSDK provides driver for the camera function using Flexible I/O.

FlexIO CAMERA driver includes 2 parts: functional APIs and eDMA transactional APIs. Functional APIs are feature/property target low level APIs. User can use functional APIs for FlexIO CAMERA initialization/configuration/operation purpose. Using the functional API require user get knowledge of the FlexIO CAMERA peripheral and know how to organize functional APIs to meet the requirement of application. All functional API use the [FLEXIO\\_CAMERA\\_Type](#) \* as the first parameter. FlexIO CAMERA functional operation groups provide the functional APIs set.

eDMA transactional APIs are transaction target high level APIs. User can use the transactional API to enable the peripheral quickly and can also use in the application if the code size and performance of transactional APIs can satisfy requirement. If the code size and performance are critical requirement, see the transactional API implementation and write their own code. All transactional APIs use the flexio\_camera\_edma\_handle\_t as the second parameter and user need to initialize the handle by calling [FLEXIO\\_CAMERA\\_TransferCreateHandleEDMA\(\)](#) API.

eDMA transactional APIs support asynchronous receive. It means, the functions [FLEXIO\\_CAMERA\\_TransferReceiveEDMA\(\)](#) set up interrupt for data receive, when the receive complete, upper layer is notified through callback function with status [kStatus\\_FLEXIO\\_CAMERA\\_RxIdle](#).

#### 15.3.2 Typical use case

##### 15.3.2.1 FlexIO CAMERA Receive using eDMA method

```
volatile uint32_t isEDMAGetOnePictureFinish = false;
edma_handle_t g_edmaHandle;
flexio_camera_edma_handle_t g_cameraEdmaHandle;
edma_config_t edmaConfig;
FLEXIO_CAMERA_Type g_FlexioCameraDevice = {.flexioBase = FLEXIO0,
                                             .datPinStartIdx = 24U, /* fxio_pin 24 -31 are used. */
                                             .pclkPinIdx = 1U,    /* fxio_pin 1 is used as pclk pin. */
                                             .hrefPinIdx = 18U,   /* flexio_pin 18 is used as href pin. */
                                             .shifterStartIdx = 0U, /* Shifter 0 = 7 are used. */
                                             .shifterCount = 8U,
                                             .timerIdx = 0U};

flexio_camera_config_t cameraConfig;

/* Configure DMAMUX */
DMAMUX_Init(DMAMUX0);
/* Configure DMA */
EDMA_GetDefaultConfig(&edmaConfig);
EDMA_Init(DMA0, &edmaConfig);

DMAMUX_SetSource(DMAMUX0, DMA_CHN_FLEXIO_TO_FRAMEBUFF, (g_FlexioCameraDevice.
                                                       shifterStartIdx + 1U));
DMAMUX_EnableChannel(DMAMUX0, DMA_CHN_FLEXIO_TO_FRAMEBUFF);
EDMA_CreateHandle(&g_edmaHandle, DMA0, DMA_CHN_FLEXIO_TO_FRAMEBUFF);

FLEXIO_CAMERA_GetDefaultConfig(&cameraConfig);
FLEXIO_CAMERA_Init(&g_FlexioCameraDevice, &cameraConfig);
/* Clear all the flag. */
```

```

FLEXIO_CAMERA_ClearStatusFlags(&g_FlexioCameraDevice,
                               kFLEXIO_CAMERA_RxDataRegFullFlag |
                               kFLEXIO_CAMERA_RxErrorFlag);
FLEXIO_ClearTimerStatusFlags(FLEXIO0, 0xFF);
FLEXIO_CAMERA_TransferCreateHandleEDMA(&g_FlexioCameraDevice, &
                                       g_cameraEdmaHandle, FLEXIO_CAMERA_UserCallback, NULL,
                                       &g_edmaHandle);
cameraTransfer.dataAddress = (uint32_t)u16CameraFrameBuffer;
cameraTransfer.dataNum = sizeof(u16CameraFrameBuffer);
FLEXIO_CAMERA_TransferReceiveEDMA(&g_FlexioCameraDevice, &
                                   g_cameraEdmaHandle, &cameraTransfer);
while (!(isEDMAGetOnePictureFinish))
{
    ;
}
/* A callback function is also needed */
void FLEXIO_CAMERA_UserCallback(FLEXIO_CAMERA_Type *base,
                                 flexio_camera_edma_handle_t *handle,
                                 status_t status,
                                 void *userData)
{
    userData = userData;
    /* eDMA Transfer finished */
    if (kStatus_FLEXIO_CAMERA_RxIdle == status)
    {
        isEDMAGetOnePictureFinish = true;
    }
}

```

## Modules

- FlexIO eDMA Camera Driver

## Data Structures

- struct **FLEXIO\_CAMERA\_Type**  
*Define structure of configuring the FlexIO camera device.* [More...](#)
- struct **flexio\_camera\_config\_t**  
*Define FlexIO camera user configuration structure.* [More...](#)
- struct **flexio\_camera\_transfer\_t**  
*Define FlexIO CAMERA transfer structure.* [More...](#)

## Macros

- #define **FLEXIO\_CAMERA\_PARALLEL\_DATA\_WIDTH** (8U)  
*Define the camera CPI interface is constantly 8-bit width.*

## Enumerations

- enum **\_flexio\_camera\_status** {
 kStatus\_FLEXIO\_CAMERA\_RxBusy = MAKE\_STATUS(kStatusGroup\_FLEXIO\_CAMERA,

## FlexIO Camera Driver

- ```
0),
kStatus_FLEXIO_CAMERA_RxIdle = MAKE_STATUS(kStatusGroup_FLEXIO_CAMERA, 1)
}

Error codes for the CAMERA driver.
• enum _flexio_camera_status_flags {
    kFLEXIO_CAMERA_RxDataRegFullFlag = 0x1U,
    kFLEXIO_CAMERA_RxErrorFlag = 0x2U }
Define FlexIO CAMERA status mask.
```

## Driver version

- #define **FSL\_FLEXIO\_CAMERA\_DRIVER\_VERSION** (MAKE\_VERSION(2, 1, 0))  
*FlexIO camera driver version 2.1.0.*

## Initialize and configuration

- void **FLEXIO\_CAMERA\_Init** (**FLEXIO\_CAMERA\_Type** \*base, const **flexio\_camera\_config\_t** \*config)  
*Ungates the FlexIO clock, reset the FlexIO module and do FlexIO CAMERA hardware configuration.*
- void **FLEXIO\_CAMERA\_Deinit** (**FLEXIO\_CAMERA\_Type** \*base)  
*Disables the FlexIO CAMERA and gate the FlexIO clock.*
- void **FLEXIO\_CAMERA\_GetDefaultConfig** (**flexio\_camera\_config\_t** \*config)  
*Get the default configuration to configure FlexIO CAMERA.*
- static void **FLEXIO\_CAMERA\_Enable** (**FLEXIO\_CAMERA\_Type** \*base, bool enable)  
*Enables/disables the FlexIO CAMERA module operation.*

## Status

- uint32\_t **FLEXIO\_CAMERA\_GetStatusFlags** (**FLEXIO\_CAMERA\_Type** \*base)  
*Gets the FlexIO CAMERA status flags.*
- void **FLEXIO\_CAMERA\_ClearStatusFlags** (**FLEXIO\_CAMERA\_Type** \*base, uint32\_t mask)  
*Clears the receive buffer full flag manually.*

## Interrupts

- void **FLEXIO\_CAMERA\_EnableInterrupt** (**FLEXIO\_CAMERA\_Type** \*base)  
*Switches on the interrupt for receive buffer full event.*
- void **FLEXIO\_CAMERA\_DisableInterrupt** (**FLEXIO\_CAMERA\_Type** \*base)  
*Switches off the interrupt for receive buffer full event.*

## DMA support

- static void **FLEXIO\_CAMERA\_EnableRxDMA** (**FLEXIO\_CAMERA\_Type** \*base, bool enable)

*Enables/disables the FlexIO CAMERA receive DMA.*

- static uint32\_t **FLEXIO\_CAMERA\_GetRxBufferAddress** (**FLEXIO\_CAMERA\_Type** \*base)  
*Gets the data from the receive buffer.*

### 15.3.3 Data Structure Documentation

#### 15.3.3.1 struct **FLEXIO\_CAMERA\_Type**

##### Data Fields

- **FLEXIO\_Type \* flexioBase**  
*FlexIO module base address.*
- **uint32\_t datPinStartIdx**  
*First data pin (D0) index for flexio\_camera.*
- **uint32\_t pclkPinIdx**  
*Pixel clock pin (PCLK) index for flexio\_camera.*
- **uint32\_t hrefPinIdx**  
*Horizontal sync pin (HREF) index for flexio\_camera.*
- **uint32\_t shifterStartIdx**  
*First shifter index used for flexio\_camera data FIFO.*
- **uint32\_t shifterCount**  
*The count of shifters that are used as flexio\_camera data FIFO.*
- **uint32\_t timerIdx**  
*Timer index used for flexio\_camera in FlexIO.*

##### 15.3.3.1.0.41 Field Documentation

###### 15.3.3.1.0.41.1 **FLEXIO\_Type\* FLEXIO\_CAMERA\_Type::flexioBase**

###### 15.3.3.1.0.41.2 **uint32\_t FLEXIO\_CAMERA\_Type::datPinStartIdx**

Then the successive following FLEXIO\_CAMERA\_DATA\_WIDTH-1 pins would be used as D1-D7.

###### 15.3.3.1.0.41.3 **uint32\_t FLEXIO\_CAMERA\_Type::pclkPinIdx**

###### 15.3.3.1.0.41.4 **uint32\_t FLEXIO\_CAMERA\_Type::hrefPinIdx**

###### 15.3.3.1.0.41.5 **uint32\_t FLEXIO\_CAMERA\_Type::shifterStartIdx**

###### 15.3.3.1.0.41.6 **uint32\_t FLEXIO\_CAMERA\_Type::shifterCount**

###### 15.3.3.1.0.41.7 **uint32\_t FLEXIO\_CAMERA\_Type::timerIdx**

#### 15.3.3.2 struct **flexio\_camera\_config\_t**

##### Data Fields

- bool **enablecamera**  
*Enable/disable FlexIO camera TX & RX.*

## FlexIO Camera Driver

- bool `enableInDoze`  
*Enable/disable FlexIO operation in doze mode.*
- bool `enableInDebug`  
*Enable/disable FlexIO operation in debug mode.*
- bool `enableFastAccess`  
*Enable/disable fast access to FlexIO registers,  
fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*

### 15.3.3.2.0.42 Field Documentation

#### 15.3.3.2.0.42.1 `bool flexio_camera_config_t::enablecamera`

#### 15.3.3.2.0.42.2 `bool flexio_camera_config_t::enableFastAccess`

### 15.3.3.3 `struct flexio_camera_transfer_t`

#### Data Fields

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

### 15.3.4 Macro Definition Documentation

#### 15.3.4.1 `#define FSL_FLEXIO_CAMERA_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

#### 15.3.4.2 `#define FLEXIO_CAMERA_PARALLEL_DATA_WIDTH (8U)`

### 15.3.5 Enumeration Type Documentation

#### 15.3.5.1 `enum _flexio_camera_status`

Enumerator

`kStatus_FLEXIO_CAMERA_RxBusy` Receiver is busy.  
`kStatus_FLEXIO_CAMERA_RxIdle` CAMERA receiver is idle.

#### 15.3.5.2 `enum _flexio_camera_status_flags`

Enumerator

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

## 15.3.6 Function Documentation

15.3.6.1 `void FLEXIO_CAMERA_Init( FLEXIO_CAMERA_Type * base, const flexio_camera_config_t * config )`

## FlexIO Camera Driver

Parameters

|               |                                                             |
|---------------|-------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_CAMERA_Type</a> structure     |
| <i>config</i> | pointer to <a href="#">flexio_camera_config_t</a> structure |

### 15.3.6.2 void FLEXIO\_CAMERA\_Deinit ( [FLEXIO\\_CAMERA\\_Type](#) \* *base* )

Note

After calling this API, user need to call [FLEXIO\\_CAMERA\\_Init](#) to use the FlexIO CAMERA module.

Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_CAMERA_Type</a> structure |
|-------------|---------------------------------------------------------|

### 15.3.6.3 void FLEXIO\_CAMERA\_GetDefaultConfig ( [flexio\\_camera\\_config\\_t](#) \* *config* )

The configuration could be used directly for calling [FLEXIO\\_CAMERA\\_Init\(\)](#). Example:

```
flexio_camera_config_t config;  
FLEXIO\_CAMERA\_GetDefaultConfig(&userConfig);
```

Parameters

|               |                                                             |
|---------------|-------------------------------------------------------------|
| <i>config</i> | pointer to <a href="#">flexio_camera_config_t</a> structure |
|---------------|-------------------------------------------------------------|

### 15.3.6.4 static void FLEXIO\_CAMERA\_Enable ( [FLEXIO\\_CAMERA\\_Type](#) \* *base*, [bool](#) *enable* ) [inline], [static]

Parameters

|               |                                               |
|---------------|-----------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_CAMERA_Type</a> |
| <i>enable</i> | True to enable, false to disable.             |

### 15.3.6.5 [uint32\\_t](#) FLEXIO\_CAMERA\_GetStatusFlags ( [FLEXIO\\_CAMERA\\_Type](#) \* *base* )

Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_CAMERA_Type</a> structure |
|-------------|---------------------------------------------------------|

Returns

FlexIO shifter status flags

- [FLEXIO\\_SHIFTSTAT\\_SSF\\_MASK](#)
- 0

#### **15.3.6.6 void FLEXIO\_CAMERA\_ClearStatusFlags ( [FLEXIO\\_CAMERA\\_Type](#) \* *base*, [uint32\\_t](#) *mask* )**

Parameters

|             |                                                                                                                                                                                                                                        |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | pointer to the device.                                                                                                                                                                                                                 |
| <i>mask</i> | status flag The parameter could be any combination of the following values: <ul style="list-style-type: none"> <li>• <a href="#">kFLEXIO_CAMERA_RxDataRegFullFlag</a></li> <li>• <a href="#">kFLEXIO_CAMERA_RxErrorFlag</a></li> </ul> |

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

Parameters

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

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

Parameters

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

#### **15.3.6.9 static void FLEXIO\_CAMERA\_EnableRxDMA ( [FLEXIO\\_CAMERA\\_Type](#) \* *base*, [bool](#) *enable* ) [inline], [static]**

## FlexIO Camera Driver

Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_CAMERA_Type</a> structure |
| <i>enable</i> | True to enable, false to disable.                       |

The FlexIO camera mode can't work without the DMA or EDMA support. Usually, it needs at least two DMA or EDMA channel, one for transferring data from camera, such as OV7670 to FlexIO buffer, another is for transferring data from FlexIO buffer to LCD.

### 15.3.6.10 static uint32\_t FLEXIO\_CAMERA\_GetRxBufferAddress ( FLEXIO\_CAMERA\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

data pointer to the buffer that would keep the data with count of *base*->shifterCount .

## 15.3.7 FlexIO eDMA Camera Driver

### 15.3.7.1 Overview

#### Data Structures

- struct `flexio_camera_edma_handle_t`  
*CAMERA eDMA handle. [More...](#)*

#### TypeDefs

- typedef void(\* `flexio_camera_edma_transfer_callback_t`)(`FLEXIO_CAMERA_Type` \*base, `flexio_camera_edma_handle_t` \*handle, `status_t` status, `void` \*userData)  
*CAMERA transfer callback function.*

#### eDMA transactional

- `status_t FLEXIO_CAMERA_TransferCreateHandleEDMA` (`FLEXIO_CAMERA_Type` \*base, `flexio_camera_edma_handle_t` \*handle, `flexio_camera_edma_transfer_callback_t` callback, `void` \*userData, `edma_handle_t` \*rxEdmaHandle)  
*Initializes the camera handle, which is used in transactional functions.*
- `status_t FLEXIO_CAMERA_TransferReceiveEDMA` (`FLEXIO_CAMERA_Type` \*base, `flexio_camera_edma_handle_t` \*handle, `flexio_camera_transfer_t` \*xfer)  
*Receives data using eDMA.*
- `void FLEXIO_CAMERA_TransferAbortReceiveEDMA` (`FLEXIO_CAMERA_Type` \*base, `flexio_camera_edma_handle_t` \*handle)  
*Aborts the receive data which used the eDMA.*
- `status_t FLEXIO_CAMERA_TransferGetReceiveCountEDMA` (`FLEXIO_CAMERA_Type` \*base, `flexio_camera_edma_handle_t` \*handle, `size_t` \*count)  
*Gets the remaining bytes to be received.*

### 15.3.7.2 Data Structure Documentation

#### 15.3.7.2.1 struct \_flexio\_camera\_edma\_handle

Forward declaration of the handle typedef.

#### Data Fields

- `flexio_camera_edma_transfer_callback_t` `callback`  
*Callback function.*
- `void *` `userData`  
*CAMERA callback function parameter.*
- `size_t` `rxSize`  
*Total bytes to be received.*
- `edma_handle_t` \* `rxEdmaHandle`

## FlexIO Camera Driver

- volatile uint8\_t **rxState**  
*RX transfer state.*

### 15.3.7.2.1.1 Field Documentation

15.3.7.2.1.1.1 **flexio\_camera\_edma\_transfer\_callback\_t flexio\_camera\_edma\_handle\_t::callback**

15.3.7.2.1.1.2 **void\* flexio\_camera\_edma\_handle\_t::userData**

15.3.7.2.1.1.3 **size\_t flexio\_camera\_edma\_handle\_t::rxSize**

15.3.7.2.1.1.4 **edma\_handle\_t\* flexio\_camera\_edma\_handle\_t::rxEdmaHandle**

### 15.3.7.3 Typedef Documentation

15.3.7.3.1 **typedef void(\* flexio\_camera\_edma\_transfer\_callback\_t)(FLEXIO\_CAMERA\_Type \*base, flexio\_camera\_edma\_handle\_t \*handle, status\_t status, void \*userData)**

### 15.3.7.4 Function Documentation

15.3.7.4.1 **status\_t FLEXIO\_CAMERA\_TransferCreateHandleEDMA ( FLEXIO\_CAMERA\_Type \* base, flexio\_camera\_edma\_handle\_t \* handle, flexio\_camera\_edma\_transfer\_callback\_t callback, void \* userData, edma\_handle\_t \* rxEdmaHandle )**

Parameters

|                     |                                                          |
|---------------------|----------------------------------------------------------|
| <i>base</i>         | pointer to <b>FLEXIO_CAMERA_Type</b> .                   |
| <i>handle</i>       | Pointer to <b>flexio_camera_edma_handle_t</b> structure. |
| <i>callback</i>     | The callback function.                                   |
| <i>userData</i>     | The parameter of the callback function.                  |
| <i>rxEdmaHandle</i> | User requested DMA handle for RX DMA transfer.           |

Return values

|                           |                                                        |
|---------------------------|--------------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                        |
| <i>kStatus_OutOfRange</i> | The FlexIO camera eDMA type/handle table out of range. |

15.3.7.4.2 **status\_t FLEXIO\_CAMERA\_TransferReceiveEDMA ( FLEXIO\_CAMERA\_Type \* base, flexio\_camera\_edma\_handle\_t \* handle, flexio\_camera\_transfer\_t \* xfer )**

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

|               |                                                                                |
|---------------|--------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_CAMERA_Type</a> .                            |
| <i>handle</i> | Pointer to the <code>flexio_camera_edma_handle_t</code> structure.             |
| <i>xfer</i>   | CAMERA eDMA transfer structure, see <a href="#">flexio_camera_transfer_t</a> . |

Return values

|                               |                              |
|-------------------------------|------------------------------|
| <i>kStatus_Success</i>        | if succeeded, others failed. |
| <i>kStatus_CAMERA_Rx-Busy</i> | Previous transfer on going.  |

#### **15.3.7.4.3 void FLEXIO\_CAMERA\_TransferAbortReceiveEDMA ( `FLEXIO_CAMERA_Type * base, flexio_camera_edma_handle_t * handle` )**

This function aborts the receive data which used the eDMA.

Parameters

|               |                                                                    |
|---------------|--------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_CAMERA_Type</a> .                |
| <i>handle</i> | Pointer to the <code>flexio_camera_edma_handle_t</code> structure. |

#### **15.3.7.4.4 status\_t FLEXIO\_CAMERA\_TransferGetReceiveCountEDMA ( `FLEXIO_CAMERA_Type * base, flexio_camera_edma_handle_t * handle, size_t * count` )**

This function gets the number of bytes still not received.

Parameters

|               |                                                                    |
|---------------|--------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_CAMERA_Type</a> .                |
| <i>handle</i> | Pointer to the <code>flexio_camera_edma_handle_t</code> structure. |
| <i>count</i>  | Number of bytes sent so far by the non-blocking transaction.       |

Return values

## FlexIO Camera Driver

|                                |                                 |
|--------------------------------|---------------------------------|
| <i>kStatus_Success</i>         | Succeed get the transfer count. |
| <i>kStatus_InvalidArgument</i> | The count parameter is invalid. |

## 15.4 FlexIO I2C Master Driver

### 15.4.1 Overview

The KSDK provides a peripheral driver for I2C master function using Flexible I/O module of Kinetis devices.

The FlexIO I2C master driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for the FlexIO I2C master initialization/configuration/operation for optimization/customization purpose. Using the functional APIs requires the knowledge of the FlexIO I2C master peripheral and how to organize functional APIs to meet the application requirements. The FlexIO I2C master functional operation groups provide the functional APIs set.

Transactional APIs 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 using the functional APIs or accessing the hardware registers.

Transactional APIs support asynchronous transfer. This means that the functions [FLEXIO\\_I2C\\_MasterTransferNonBlocking\(\)](#) set up the interrupt non-blocking transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus\_Success status.

### 15.4.2 Typical use case

#### 15.4.2.1 FlexIO I2C master transfer using an interrupt method

```
flexio_i2c_master_handle_t g_m_handle;
flexio_i2c_master_config_t masterConfig;
flexio_i2c_master_transfer_t masterXfer;
volatile bool completionFlag = false;
const uint8_t sendData[] = [.....];
FLEXIO_I2C_Type i2cDev;

void FLEXIO_I2C_MasterCallback(FLEXIO_I2C_Type *base, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_Success == status)
    {
        completionFlag = true;
    }
}

void main(void)
{
    //...

    FLEXIO_I2C_MasterGetDefaultConfig(&masterConfig);

    FLEXIO_I2C_MasterInit(&i2cDev, &user_config);
    FLEXIO_I2C_MasterTransferCreateHandle(&i2cDev, &g_m_handle,
   FLEXIO_I2C_MasterCallback, NULL);

    // Prepares to send.
```

## FlexIO I2C Master Driver

```
masterXfer.slaveAddress = g_accel_address[0];
masterXfer.direction = kI2C_Read;
masterXfer.subaddress = &who_am_i_reg;
masterXfer.subaddressSize = 1;
masterXfer.data = &who_am_i_value;
masterXfer.dataSize = 1;
masterXfer.flags = kI2C_TransferDefaultFlag;

// Sends out.
FLEXIO_I2C_MasterTransferNonBlocking(&i2cDev, &g_m_handle, &
    masterXfer);

// Wait for sending is complete.
while (!completionFlag)
{
}

// ...
}
```

## Data Structures

- struct **FLEXIO\_I2C\_Type**  
*Define FlexIO I2C master access structure typedef.* [More...](#)
- struct **flexio\_i2c\_master\_config\_t**  
*Define FlexIO I2C master user configuration structure.* [More...](#)
- struct **flexio\_i2c\_master\_transfer\_t**  
*Define FlexIO I2C master transfer structure.* [More...](#)
- struct **flexio\_i2c\_master\_handle\_t**  
*Define FlexIO I2C master handle structure.* [More...](#)

## TypeDefs

- typedef void(\* **flexio\_i2c\_master\_transfer\_callback\_t** )(FLEXIO\_I2C\_Type \*base, flexio\_i2c\_master\_handle\_t \*handle, status\_t status, void \*userData)  
*FlexIO I2C master transfer callback typedef.*

## Enumerations

- enum **\_flexio\_i2c\_status** {  
 kStatus\_FLEXIO\_I2C\_Busy = MAKE\_STATUS(kStatusGroup\_FLEXIO\_I2C, 0),  
 kStatus\_FLEXIO\_I2C\_Idle = MAKE\_STATUS(kStatusGroup\_FLEXIO\_I2C, 1),  
 kStatus\_FLEXIO\_I2C\_Nak = MAKE\_STATUS(kStatusGroup\_FLEXIO\_I2C, 2) }  
*FlexIO I2C transfer status.*
- enum **\_flexio\_i2c\_master\_interrupt** {  
 kFLEXIO\_I2C\_TxEmptyInterruptEnable = 0x1U,  
 kFLEXIO\_I2C\_RxFullInterruptEnable = 0x2U }  
*Define FlexIO I2C master interrupt mask.*
- enum **\_flexio\_i2c\_master\_status\_flags** {

```
kFLEXIO_I2C_TxEmptyFlag = 0x1U,
kFLEXIO_I2C_RxFullFlag = 0x2U,
kFLEXIO_I2C_ReceiveNakFlag = 0x4U }
```

*Define FlexIO I2C master status mask.*

- enum `flexio_i2c_direction_t` {
   
kFLEXIO\_I2C\_Write = 0x0U,
   
kFLEXIO\_I2C\_Read = 0x1U }

*Direction of master transfer.*

## Driver version

- #define `FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 2)`)
   
*FlexIO I2C master driver version 2.1.2.*

## Initialization and deinitialization

- void `FLEXIO_I2C_MasterInit` (`FLEXIO_I2C_Type` \*base, `flexio_i2c_master_config_t` \*masterConfig, `uint32_t` srcClock\_Hz)
   
*Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.*
- void `FLEXIO_I2C_MasterDeinit` (`FLEXIO_I2C_Type` \*base)
   
*De-initializes the FlexIO I2C master peripheral.*
- void `FLEXIO_I2C_MasterGetDefaultConfig` (`flexio_i2c_master_config_t` \*masterConfig)
   
*Gets the default configuration to configure the FlexIO module.*
- static void `FLEXIO_I2C_MasterEnable` (`FLEXIO_I2C_Type` \*base, `bool` enable)
   
*Enables/disables the FlexIO module operation.*

## Status

- `uint32_t FLEXIO_I2C_MasterGetStatusFlags` (`FLEXIO_I2C_Type` \*base)
   
*Gets the FlexIO I2C master status flags.*
- void `FLEXIO_I2C_MasterClearStatusFlags` (`FLEXIO_I2C_Type` \*base, `uint32_t` mask)
   
*Clears the FlexIO I2C master status flags.*

## Interrupts

- void `FLEXIO_I2C_MasterEnableInterrupts` (`FLEXIO_I2C_Type` \*base, `uint32_t` mask)
   
*Enables the FlexIO i2c master interrupt requests.*
- void `FLEXIO_I2C_MasterDisableInterrupts` (`FLEXIO_I2C_Type` \*base, `uint32_t` mask)
   
*Disables the FlexIO I2C master interrupt requests.*

## FlexIO I2C Master Driver

### Bus Operations

- void `FLEXIO_I2C_MasterSetBaudRate` (`FLEXIO_I2C_Type` \*base, `uint32_t` baudRate\_Bps, `uint32_t` srcClock\_Hz)  
*Sets the FlexIO I2C master transfer baudrate.*
- void `FLEXIO_I2C_MasterStart` (`FLEXIO_I2C_Type` \*base, `uint8_t` address, `flexio_i2c_direction_t` direction)  
*Sends START + 7-bit address to the bus.*
- void `FLEXIO_I2C_MasterStop` (`FLEXIO_I2C_Type` \*base)  
*Sends the stop signal on the bus.*
- void `FLEXIO_I2C_MasterRepeatedStart` (`FLEXIO_I2C_Type` \*base)  
*Sends the repeated start signal on the bus.*
- void `FLEXIO_I2C_MasterAbortStop` (`FLEXIO_I2C_Type` \*base)  
*Sends the stop signal when transfer is still on-going.*
- void `FLEXIO_I2C_MasterEnableAck` (`FLEXIO_I2C_Type` \*base, `bool` enable)  
*Configures the sent ACK/NAK for the following byte.*
- `status_t FLEXIO_I2C_MasterSetTransferCount` (`FLEXIO_I2C_Type` \*base, `uint8_t` count)  
*Sets the number of bytes to be transferred from a start signal to a stop signal.*
- static void `FLEXIO_I2C_MasterWriteByte` (`FLEXIO_I2C_Type` \*base, `uint32_t` data)  
*Writes one byte of data to the I2C bus.*
- static `uint8_t FLEXIO_I2C_MasterReadByte` (`FLEXIO_I2C_Type` \*base)  
*Reads one byte of data from the I2C bus.*
- `status_t FLEXIO_I2C_MasterWriteBlocking` (`FLEXIO_I2C_Type` \*base, `const uint8_t` \*txBuff, `uint8_t` txSize)  
*Sends a buffer of data in bytes.*
- void `FLEXIO_I2C_MasterReadBlocking` (`FLEXIO_I2C_Type` \*base, `uint8_t` \*rxBuff, `uint8_t` rxSize)  
*Receives a buffer of bytes.*
- `status_t FLEXIO_I2C_MasterTransferBlocking` (`FLEXIO_I2C_Type` \*base, `flexio_i2c_master_transfer_t` \*xfer)  
*Performs a master polling transfer on the I2C bus.*

### Transactional

- `status_t FLEXIO_I2C_MasterTransferCreateHandle` (`FLEXIO_I2C_Type` \*base, `flexio_i2c_master_handle_t` \*handle, `flexio_i2c_master_transfer_callback_t` callback, `void` \*userData)  
*Initializes the I2C handle which is used in transactional functions.*
- `status_t FLEXIO_I2C_MasterTransferNonBlocking` (`FLEXIO_I2C_Type` \*base, `flexio_i2c_master_handle_t` \*handle, `flexio_i2c_master_transfer_t` \*xfer)  
*Performs a master interrupt non-blocking transfer on the I2C bus.*
- `status_t FLEXIO_I2C_MasterTransferGetCount` (`FLEXIO_I2C_Type` \*base, `flexio_i2c_master_handle_t` \*handle, `size_t` \*count)  
*Gets the master transfer status during a interrupt non-blocking transfer.*
- void `FLEXIO_I2C_MasterTransferAbort` (`FLEXIO_I2C_Type` \*base, `flexio_i2c_master_handle_t` \*handle)  
*Aborts an interrupt non-blocking transfer early.*
- void `FLEXIO_I2C_MasterTransferHandleIRQ` (`void` \*i2cType, `void` \*i2cHandle)  
*Master interrupt handler.*

## 15.4.3 Data Structure Documentation

### 15.4.3.1 struct FLEXIO\_I2C\_Type

#### Data Fields

- `FLEXIO_Type * flexioBase`  
*FlexIO base pointer.*
- `uint8_t SDAPinIndex`  
*Pin select for I2C SDA.*
- `uint8_t SCLPinIndex`  
*Pin select for I2C SCL.*
- `uint8_t shifterIndex [2]`  
*Shifter index used in FlexIO I2C.*
- `uint8_t timerIndex [2]`  
*Timer index used in FlexIO I2C.*

#### 15.4.3.1.0.1 Field Documentation

##### 15.4.3.1.0.1.1 `FLEXIO_Type* FLEXIO_I2C_Type::flexioBase`

##### 15.4.3.1.0.1.2 `uint8_t FLEXIO_I2C_Type::SDAPinIndex`

##### 15.4.3.1.0.1.3 `uint8_t FLEXIO_I2C_Type::SCLPinIndex`

##### 15.4.3.1.0.1.4 `uint8_t FLEXIO_I2C_Type::shifterIndex[2]`

##### 15.4.3.1.0.1.5 `uint8_t FLEXIO_I2C_Type::timerIndex[2]`

### 15.4.3.2 struct flexio\_i2c\_master\_config\_t

#### Data Fields

- `bool enableMaster`  
*Enables the FLEXIO I2C peripheral at initialization time.*
- `bool enableInDoze`  
*Enable/disable FlexIO operation in doze mode.*
- `bool enableInDebug`  
*Enable/disable FlexIO operation in debug mode.*
- `bool enableFastAccess`  
*Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- `uint32_t baudRate_Bps`  
*Baud rate in Bps.*

## FlexIO I2C Master Driver

### 15.4.3.2.0.2 Field Documentation

- 15.4.3.2.0.2.1 `bool flexio_i2c_master_config_t::enableMaster`
- 15.4.3.2.0.2.2 `bool flexio_i2c_master_config_t::enableInDoze`
- 15.4.3.2.0.2.3 `bool flexio_i2c_master_config_t::enableInDebug`
- 15.4.3.2.0.2.4 `bool flexio_i2c_master_config_t::enableFastAccess`
- 15.4.3.2.0.2.5 `uint32_t flexio_i2c_master_config_t::baudRate_Bps`

### 15.4.3.3 struct flexio\_i2c\_master\_transfer\_t

#### Data Fields

- `uint32_t flags`  
*Transfer flag which controls the transfer, reserved for FlexIO I2C.*
- `uint8_t slaveAddress`  
*7-bit slave address.*
- `flexio_i2c_direction_t direction`  
*Transfer direction, read or write.*
- `uint32_t subaddress`  
*Sub address.*
- `uint8_t subaddressSize`  
*Size of command buffer.*
- `uint8_t volatile * data`  
*Transfer buffer.*
- `volatile size_t dataSize`  
*Transfer size.*

### 15.4.3.3.0.3 Field Documentation

- 15.4.3.3.0.3.1 `uint32_t flexio_i2c_master_transfer_t::flags`
- 15.4.3.3.0.3.2 `uint8_t flexio_i2c_master_transfer_t::slaveAddress`
- 15.4.3.3.0.3.3 `flexio_i2c_direction_t flexio_i2c_master_transfer_t::direction`
- 15.4.3.3.0.3.4 `uint32_t flexio_i2c_master_transfer_t::subaddress`

Transferred MSB first.

**15.4.3.3.0.3.5** `uint8_t flexio_i2c_master_transfer_t::subaddressSize`

**15.4.3.3.0.3.6** `uint8_t volatile* flexio_i2c_master_transfer_t::data`

**15.4.3.3.0.3.7** `volatile size_t flexio_i2c_master_transfer_t::dataSize`

#### 15.4.3.4 `struct _flexio_i2c_master_handle`

FlexIO I2C master handle typedef.

#### Data Fields

- `flexio_i2c_master_transfer_t transfer`  
*FlexIO I2C master transfer copy.*
- `size_t transferSize`  
*Total bytes to be transferred.*
- `uint8_t state`  
*Transfer state maintained during transfer.*
- `flexio_i2c_master_transfer_callback_t completionCallback`  
*Callback function called at transfer event.*
- `void *userData`  
*Callback parameter passed to callback function.*

#### 15.4.3.4.0.4 Field Documentation

**15.4.3.4.0.4.1** `flexio_i2c_master_transfer_t flexio_i2c_master_handle_t::transfer`

**15.4.3.4.0.4.2** `size_t flexio_i2c_master_handle_t::transferSize`

**15.4.3.4.0.4.3** `uint8_t flexio_i2c_master_handle_t::state`

**15.4.3.4.0.4.4** `flexio_i2c_master_transfer_callback_t flexio_i2c_master_handle_t::completionCallback`

Callback function called at transfer event.

## FlexIO I2C Master Driver

15.4.3.4.0.4.5 `void* flexio_i2c_master_handle_t::userData`

### 15.4.4 Macro Definition Documentation

15.4.4.1 `#define FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))`

### 15.4.5 Typedef Documentation

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

### 15.4.6 Enumeration Type Documentation

#### 15.4.6.1 enum \_flexio\_i2c\_status

Enumerator

*kStatus\_FLEXIO\_I2C\_Busy* I2C is busy doing transfer.

*kStatus\_FLEXIO\_I2C\_Idle* I2C is busy doing transfer.

*kStatus\_FLEXIO\_I2C\_Nak* NAK received during transfer.

#### 15.4.6.2 enum \_flexio\_i2c\_master\_interrupt

Enumerator

*kFLEXIO\_I2C\_TxEmptyInterruptEnable* Tx buffer empty interrupt enable.

*kFLEXIO\_I2C\_RxFullInterruptEnable* Rx buffer full interrupt enable.

#### 15.4.6.3 enum \_flexio\_i2c\_master\_status\_flags

Enumerator

*kFLEXIO\_I2C\_TxEmptyFlag* Tx shifter empty flag.

*kFLEXIO\_I2C\_RxFullFlag* Rx shifter full/Transfer complete flag.

*kFLEXIO\_I2C\_ReceiveNakFlag* Receive NAK flag.

#### 15.4.6.4 enum flexio\_i2c\_direction\_t

Enumerator

*kFLEXIO\_I2C\_Write* Master send to slave.

*kFLEXIO\_I2C\_Read* Master receive from slave.

## 15.4.7 Function Documentation

### 15.4.7.1 void FLEXIO\_I2C\_MasterInit ( **FLEXIO\_I2C\_Type** \* *base*, **flexio\_i2c\_master\_config\_t** \* *masterConfig*, **uint32\_t** *srcClock\_Hz* )

Example

```
FLEXIO_I2C_Type base = {
    .flexioBase = FLEXIO,
    .SDAPinIndex = 0,
    .SCLPinIndex = 1,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_i2c_master_config_t config = {
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 100000
};
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

Parameters

|                     |                                                         |
|---------------------|---------------------------------------------------------|
| <i>base</i>         | pointer to <b>FLEXIO_I2C_Type</b> structure.            |
| <i>masterConfig</i> | pointer to <b>flexio_i2c_master_config_t</b> structure. |
| <i>srcClock_Hz</i>  | FlexIO source clock in Hz.                              |

### 15.4.7.2 void FLEXIO\_I2C\_MasterDeinit ( **FLEXIO\_I2C\_Type** \* *base* )

Calling this API gates the FlexIO clock, so the FlexIO I2C master module can't work unless call **FLEXIO\_I2C\_MasterInit**.

Parameters

|             |                                              |
|-------------|----------------------------------------------|
| <i>base</i> | pointer to <b>FLEXIO_I2C_Type</b> structure. |
|-------------|----------------------------------------------|

### 15.4.7.3 void FLEXIO\_I2C\_MasterGetDefaultConfig ( **flexio\_i2c\_master\_config\_t** \* *masterConfig* )

The configuration can be used directly for calling **FLEXIO\_I2C\_MasterInit()**.

Example:

```
flexio_i2c_master_config_t config;
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

## FlexIO I2C Master Driver

Parameters

|                     |                                                                  |
|---------------------|------------------------------------------------------------------|
| <i>masterConfig</i> | pointer to <a href="#">flexio_i2c_master_config_t</a> structure. |
|---------------------|------------------------------------------------------------------|

**15.4.7.4 static void FLEXIO\_I2C\_MasterEnable ( **FLEXIO\_I2C\_Type** \* *base*, **bool enable** ) [inline], [static]**

Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
| <i>enable</i> | pass true to enable module, false to disable module.  |

**15.4.7.5 uint32\_t FLEXIO\_I2C\_MasterGetStatusFlags ( **FLEXIO\_I2C\_Type** \* *base* )**

Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2C_Type</a> structure |
|-------------|------------------------------------------------------|

Returns

status flag, use status flag to AND [\\_flexio\\_i2c\\_master\\_status\\_flags](#) could get the related status.

**15.4.7.6 void FLEXIO\_I2C\_MasterClearStatusFlags ( **FLEXIO\_I2C\_Type** \* *base*, **uint32\_t mask** )**

Parameters

|             |                                                                                                                                                                                            |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2C_Type</a> structure.                                                                                                                                      |
| <i>mask</i> | status flag. The parameter could be any combination of the following values: <ul style="list-style-type: none"><li>• kFLEXIO_I2C_RxFullFlag</li><li>• kFLEXIO_I2C_ReceiveNakFlag</li></ul> |

**15.4.7.7 void FLEXIO\_I2C\_MasterEnableInterrupts ( **FLEXIO\_I2C\_Type** \* *base*, **uint32\_t mask** )**

Parameters

|             |                                                                                                                                                              |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2C_Type</a> structure.                                                                                                        |
| <i>mask</i> | interrupt source. Currently only one interrupt request source: <ul style="list-style-type: none"><li>• kFLEXIO_I2C_TransferCompleteInterruptEnable</li></ul> |

#### 15.4.7.8 void FLEXIO\_I2C\_MasterDisableInterrupts ( [FLEXIO\\_I2C\\_Type](#) \* *base*, [uint32\\_t](#) *mask* )

Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
| <i>mask</i> | interrupt source.                                     |

#### 15.4.7.9 void FLEXIO\_I2C\_MasterSetBaudRate ( [FLEXIO\\_I2C\\_Type](#) \* *base*, [uint32\\_t](#) *baudRate\_Bps*, [uint32\\_t](#) *srcClock\_Hz* )

Parameters

|                     |                                                      |
|---------------------|------------------------------------------------------|
| <i>base</i>         | pointer to <a href="#">FLEXIO_I2C_Type</a> structure |
| <i>baudRate_Bps</i> | the baud rate value in HZ                            |
| <i>srcClock_Hz</i>  | source clock in HZ                                   |

#### 15.4.7.10 void FLEXIO\_I2C\_MasterStart ( [FLEXIO\\_I2C\\_Type](#) \* *base*, [uint8\\_t](#) *address*, [flexio\\_i2c\\_direction\\_t](#) *direction* )

Note

This API should be called when transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but not address transfer finished on the bus. Ensure that the kFLEXIO\_I2C\_RxFullFlag status is asserted before calling this API.

## FlexIO I2C Master Driver

Parameters

|                  |                                                                                                                                                                                                                                              |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | pointer to <a href="#">FLEXIO_I2C_Type</a> structure.                                                                                                                                                                                        |
| <i>address</i>   | 7-bit address.                                                                                                                                                                                                                               |
| <i>direction</i> | transfer direction. This parameter is one of the values in <code>flexio_i2c_direction_t</code> : <ul style="list-style-type: none"><li>• <code>kFLEXIO_I2C_Write</code>: Transmit</li><li>• <code>kFLEXIO_I2C_Read</code>: Receive</li></ul> |

### 15.4.7.11 void FLEXIO\_I2C\_MasterStop ( [FLEXIO\\_I2C\\_Type](#) \* *base* )

Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
|-------------|-------------------------------------------------------|

### 15.4.7.12 void FLEXIO\_I2C\_MasterRepeatedStart ( [FLEXIO\\_I2C\\_Type](#) \* *base* )

Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
|-------------|-------------------------------------------------------|

### 15.4.7.13 void FLEXIO\_I2C\_MasterAbortStop ( [FLEXIO\\_I2C\\_Type](#) \* *base* )

Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
|-------------|-------------------------------------------------------|

### 15.4.7.14 void FLEXIO\_I2C\_MasterEnableAck ( [FLEXIO\\_I2C\\_Type](#) \* *base*, `bool enable` )

Parameters

|               |                                                          |
|---------------|----------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_I2C_Type</a> structure.    |
| <i>enable</i> | true to configure send ACK, false configure to send NAK. |

### 15.4.7.15 `status_t` FLEXIO\_I2C\_MasterSetTransferCount ( [FLEXIO\\_I2C\\_Type](#) \* *base*, `uint8_t count` )

## Note

Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

## Parameters

|              |                                                                                      |
|--------------|--------------------------------------------------------------------------------------|
| <i>base</i>  | pointer to <a href="#">FLEXIO_I2C_Type</a> structure.                                |
| <i>count</i> | number of bytes need to be transferred from a start signal to a re-start/stop signal |

## Return values

|                                |                                    |
|--------------------------------|------------------------------------|
| <i>kStatus_Success</i>         | Successfully configured the count. |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.         |

**15.4.7.16 static void FLEXIO\_I2C\_MasterWriteByte ( [FLEXIO\\_I2C\\_Type](#) \* *base*, [uint32\\_t](#) *data* ) [inline], [static]**

## Note

This is a non-blocking API, which returns directly after the data is put into the data register but not data transfer finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

## Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
| <i>data</i> | a byte of data.                                       |

**15.4.7.17 static [uint8\\_t](#) FLEXIO\_I2C\_MasterReadByte ( [FLEXIO\\_I2C\\_Type](#) \* *base* ) [inline], [static]**

## Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

## Parameters

## FlexIO I2C Master Driver

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
|-------------|-------------------------------------------------------|

Returns

data byte read.

### 15.4.7.18 **status\_t FLEXIO\_I2C\_MasterWriteBlocking ( [FLEXIO\\_I2C\\_Type](#) \* *base*, const [uint8\\_t](#) \* *txBuff*, [uint8\\_t](#) *txSize* )**

Note

This function blocks via polling until all bytes have been sent.

Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
| <i>txBuff</i> | The data bytes to send.                               |
| <i>txSize</i> | The number of data bytes to send.                     |

Return values

|                               |                                  |
|-------------------------------|----------------------------------|
| <i>kStatus_Success</i>        | Successfully write data.         |
| <i>kStatus_FLEXIO_I2C_Nak</i> | Receive NAK during writing data. |

### 15.4.7.19 **void FLEXIO\_I2C\_MasterReadBlocking ( [FLEXIO\\_I2C\\_Type](#) \* *base*, [uint8\\_t](#) \* *rxBuff*, [uint8\\_t](#) *rxSize* )**

Note

This function blocks via polling until all bytes have been received.

Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
| <i>rxBuff</i> | The buffer to store the received bytes.               |
| <i>rxSize</i> | The number of data bytes to be received.              |

#### 15.4.7.20 status\_t FLEXIO\_I2C\_MasterTransferBlocking ( **FLEXIO\_I2C\_Type \* base,** **flexio\_i2c\_master\_transfer\_t \* xfer** )

Note

The API does not return until the transfer succeeds or fails due to receiving NAK.

Parameters

|             |                                                                    |
|-------------|--------------------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2C_Type</a> structure.              |
| <i>xfer</i> | pointer to <a href="#">flexio_i2c_master_transfer_t</a> structure. |

Returns

status of [status\\_t](#).

#### 15.4.7.21 status\_t FLEXIO\_I2C\_MasterTransferCreateHandle ( **FLEXIO\_I2C\_Type \* base,** **flexio\_i2c\_master\_handle\_t \* handle,** **flexio\_i2c\_master\_transfer\_callback\_t** **callback,** **void \* userData** )

Parameters

|                 |                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------|
| <i>base</i>     | pointer to <a href="#">FLEXIO_I2C_Type</a> structure.                                        |
| <i>handle</i>   | pointer to <a href="#">flexio_i2c_master_handle_t</a> structure to store the transfer state. |
| <i>callback</i> | pointer to user callback function.                                                           |
| <i>userData</i> | user param passed to the callback function.                                                  |

Return values

|                           |                                                |
|---------------------------|------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                |
| <i>kStatus_OutOfRange</i> | The FlexIO type/handle/isr table out of range. |

#### 15.4.7.22 status\_t FLEXIO\_I2C\_MasterTransferNonBlocking ( **FLEXIO\_I2C\_Type \* base,** **flexio\_i2c\_master\_handle\_t \* handle,** **flexio\_i2c\_master\_transfer\_t \* xfer** )

Note

The API returns immediately after the transfer initiates. Call [FLEXIO\\_I2C\\_MasterGetTransferCount](#) to poll the transfer status to check whether the transfer is finished. If the return status is not [kStatus\\_FLEXIO\\_I2C\\_Busy](#), the transfer is finished.

## FlexIO I2C Master Driver

Parameters

|               |                                                                                                 |
|---------------|-------------------------------------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_I2C_Type</a> structure                                            |
| <i>handle</i> | pointer to <a href="#">flexio_i2c_master_handle_t</a> structure which stores the transfer state |
| <i>xfer</i>   | pointer to <a href="#">flexio_i2c_master_transfer_t</a> structure                               |

Return values

|                                |                                                      |
|--------------------------------|------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                       |
| <i>kStatus_FLEXIO_I2C_Busy</i> | FlexIO I2C is not idle, is running another transfer. |

### 15.4.7.23 **status\_t FLEXIO\_I2C\_MasterTransferGetCount ( [FLEXIO\\_I2C\\_Type](#) \* *base*, [flexio\\_i2c\\_master\\_handle\\_t](#) \* *handle*, [size\\_t](#) \* *count* )**

Parameters

|               |                                                                                                  |
|---------------|--------------------------------------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_I2C_Type</a> structure.                                            |
| <i>handle</i> | pointer to <a href="#">flexio_i2c_master_handle_t</a> 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. |

### 15.4.7.24 **void FLEXIO\_I2C\_MasterTransferAbort ( [FLEXIO\\_I2C\\_Type](#) \* *base*, [flexio\\_i2c\\_master\\_handle\\_t](#) \* *handle* )**

Note

This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

|               |                                                                                                 |
|---------------|-------------------------------------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_I2C_Type</a> structure                                            |
| <i>handle</i> | pointer to <a href="#">flexio_i2c_master_handle_t</a> structure which stores the transfer state |

15.4.7.25 **void FLEXIO\_I2C\_MasterTransferHandleIRQ ( void \* *i2cType*, void \* *i2cHandle* )**

## FlexIO I2C Master Driver

Parameters

|                  |                                                                   |
|------------------|-------------------------------------------------------------------|
| <i>i2cType</i>   | pointer to <a href="#">FLEXIO_I2C_Type</a> structure              |
| <i>i2cHandle</i> | pointer to <a href="#">flexio_i2c_master_transfer_t</a> structure |

## 15.5 FlexIO I2S Driver

### 15.5.1 Overview

The KSDK provides a peripheral driver for I2S function using Flexible I/O module of Kinetis devices.

The FlexIO I2S driver includes functional APIs and transactional APIs.

Functional APIs target low level APIs. Functional APIs can be used for FlexIO I2S initialization/configuration/operation for optimization/customization purpose. Using the functional API requires knowledge of the FlexIO I2S peripheral and how to organize functional APIs to meet the application requirements. All functional APIs use the peripheral base address as the first parameter. FlexIO I2S functional operation groups provide the functional APIs set.

Transactional APIs target high level APIs. The transactional APIs can be used to enable the peripheral and can also be used 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. All transactional APIs use the the `sai_handle_t` as the first parameter. Initialize the handle by calling the `FlexIO_I2S_TransferTxCreateHandle()` or `FlexIO_I2S_TransferRxCreateHandle()` API.

Transactional APIs support asynchronous transfer. This means that the functions `FLEXIO_I2S_TransferSendNonBlocking()` and `FLEXIO_I2S_TransferReceiveNonBlocking()` set up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_FLEXIO_I2S_TxIdle` and `kStatus_FLEXIO_I2S_RxIdle` status.

### 15.5.2 Typical use case

#### 15.5.2.1 FlexIO I2S send/receive using an interrupt method

```

sai_handle_t g_saiTxHandle;
sai_config_t user_config;
sai_transfer_t sendXfer;
volatile bool txFinished;
volatile bool rxFinished;
const uint8_t sendData[] = [.....];

void FLEXIO_I2S_UserCallback(sai_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_I2S_TxIdle == status)
    {
        txFinished = true;
    }
}

void main(void)
{
    //...

    FLEXIO_I2S_TxGetDefaultConfig(&user_config);

    FLEXIO_I2S_TxInit(FLEXIO_I2S0, &user_config);
    FLEXIO_I2S_TransferTxCreateHandle(FLEXIO_I2S0, &g_saiHandle,

```

## FlexIO I2S Driver

```
FLEXIO_I2S_UserCallback, NULL);

//Configures the SAI format.
FLEXIO_I2S_TransferTxSetTransferFormat(FLEXIO_I2S0, &g_saiHandle, mclkSource, mclk);

// Prepares to send.
sendXfer.data = sendData
sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
txFinished = false;

// Sends out.
FLEXIO_I2S_TransferSendNonBlocking(FLEXIO_I2S0, &g_saiHandle, &
sendXfer);

// Waiting to send is finished.
while (!txFinished)
{
}

// ...
}
```

### 15.5.2.2 FLEXIO\_I2S send/receive using a DMA method

```
sai_handle_t g_saiHandle;
dma_handle_t g_saiTxDmaHandle;
dma_handle_t g_saiRxDmaHandle;
sai_config_t user_config;
sai_transfer_t sendXfer;
volatile bool txFinished;
uint8_t sendData[] = ...;

void FLEXIO_I2S_UserCallback(sai_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_I2S_TxIdle == status)
    {
        txFinished = true;
    }
}

void main(void)
{
    //...

    FLEXIO_I2S_TxGetDefaultConfig(&user_config);
    FLEXIO_I2S_TxInit(FLEXIO_I2S0, &user_config);

    // Sets up the DMA.
    DMAMUX_Init(DMAMUX0);
    DMAMUX_SetSource(DMAMUX0, FLEXIO_I2S_TX_DMA_CHANNEL, FLEXIO_I2S_TX_DMA_REQUEST);
    DMAMUX_EnableChannel(DMAMUX0, FLEXIO_I2S_TX_DMA_CHANNEL);

    DMA_Init(DMA0);

    /* Creates the DMA handle. */
    DMA_TransferTxCreateHandle(&g_saiTxDmaHandle, DMA0, FLEXIO_I2S_TX_DMA_CHANNEL);

    FLEXIO_I2S_TransferTxCreateHandleDMA(FLEXIO_I2S0, &g_saiTxDmaHandle
        , FLEXIO_I2S_UserCallback, NULL);

    // Prepares to send.
    sendXfer.data = sendData
    sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
```

```

txFinished = false;

// Sends out.
FLEXIO_I2S_TransferSendDMA(&g_saiHandle, &sendXfer);

// Waiting to send is finished.
while (!txFinished)
{
}

// ...
}

```

## Modules

- FlexIO DMA I2S Driver
- FlexIO eDMA I2S Driver

## Data Structures

- struct **FLEXIO\_I2S\_Type**  
*Define FlexIO I2S access structure typedef.* [More...](#)
- struct **flexio\_i2s\_config\_t**  
*FlexIO I2S configure structure.* [More...](#)
- struct **flexio\_i2s\_format\_t**  
*FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.* [More...](#)
- struct **flexio\_i2s\_transfer\_t**  
*Define FlexIO I2S transfer structure.* [More...](#)
- struct **flexio\_i2s\_handle\_t**  
*Define FlexIO I2S handle structure.* [More...](#)

## Macros

- #define **FLEXIO\_I2S\_XFER\_QUEUE\_SIZE** (4)  
*FlexIO I2S transfer queue size, user can refine it according to use case.*

## Typedefs

- typedef void(\* **flexio\_i2s\_callback\_t** )(FLEXIO\_I2S\_Type \*base, flexio\_i2s\_handle\_t \*handle, status\_t status, void \*userData)  
*FlexIO I2S xfer callback prototype.*

### Enumerations

- enum `_flexio_i2s_status` {  
  `kStatus_FLEXIO_I2S_Idle` = MAKE\_STATUS(kStatusGroup\_FLEXIO\_I2S, 0),  
  `kStatus_FLEXIO_I2S_TxBusy` = MAKE\_STATUS(kStatusGroup\_FLEXIO\_I2S, 1),  
  `kStatus_FLEXIO_I2S_RxBusy` = MAKE\_STATUS(kStatusGroup\_FLEXIO\_I2S, 2),  
  `kStatus_FLEXIO_I2S_Error` = MAKE\_STATUS(kStatusGroup\_FLEXIO\_I2S, 3),  
  `kStatus_FLEXIO_I2S_QueueFull` = MAKE\_STATUS(kStatusGroup\_FLEXIO\_I2S, 4) }  
    *FlexIO I2S transfer status.*
- enum `flexio_i2s_master_slave_t` {  
  `kFLEXIO_I2S_Master` = 0x0U,  
  `kFLEXIO_I2S_Slave` = 0x1U }  
    *Master or slave mode.*
- enum `_flexio_i2s_interrupt_enable` {  
  `kFLEXIO_I2S_TxDataRegEmptyInterruptEnable` = 0x1U,  
  `kFLEXIO_I2S_RxDataRegFullInterruptEnable` = 0x2U }  
    *Define FlexIO FlexIO I2S interrupt mask.*
- enum `_flexio_i2s_status_flags` {  
  `kFLEXIO_I2S_TxDataRegEmptyFlag` = 0x1U,  
  `kFLEXIO_I2S_RxDataRegFullFlag` = 0x2U }  
    *Define FlexIO FlexIO I2S status mask.*
- enum `flexio_i2s_sample_rate_t` {  
  `kFLEXIO_I2S_SampleRate8KHz` = 8000U,  
  `kFLEXIO_I2S_SampleRate11025Hz` = 11025U,  
  `kFLEXIO_I2S_SampleRate12KHz` = 12000U,  
  `kFLEXIO_I2S_SampleRate16KHz` = 16000U,  
  `kFLEXIO_I2S_SampleRate22050Hz` = 22050U,  
  `kFLEXIO_I2S_SampleRate24KHz` = 24000U,  
  `kFLEXIO_I2S_SampleRate32KHz` = 32000U,  
  `kFLEXIO_I2S_SampleRate44100Hz` = 44100U,  
  `kFLEXIO_I2S_SampleRate48KHz` = 48000U,  
  `kFLEXIO_I2S_SampleRate96KHz` = 96000U }  
    *Audio sample rate.*
- enum `flexio_i2s_word_width_t` {  
  `kFLEXIO_I2S_WordWidth8bits` = 8U,  
  `kFLEXIO_I2S_WordWidth16bits` = 16U,  
  `kFLEXIO_I2S_WordWidth24bits` = 24U,  
  `kFLEXIO_I2S_WordWidth32bits` = 32U }  
    *Audio word width.*

### Driver version

- #define `FSL_FLEXIO_I2S_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 1))  
    *FlexIO I2S driver version 2.1.0.*

## Initialization and deinitialization

- void **FLEXIO\_I2S\_Init** (**FLEXIO\_I2S\_Type** \*base, const **flexio\_i2s\_config\_t** \*config)  
*Initializes the FlexIO I2S.*
- void **FLEXIO\_I2S\_GetDefaultConfig** (**flexio\_i2s\_config\_t** \*config)  
*Sets the FlexIO I2S configuration structure to default values.*
- void **FLEXIO\_I2S\_Deinit** (**FLEXIO\_I2S\_Type** \*base)  
*De-initializes the FlexIO I2S.*
- static void **FLEXIO\_I2S\_Enable** (**FLEXIO\_I2S\_Type** \*base, bool enable)  
*Enables/disables the FlexIO I2S module operation.*

## Status

- uint32\_t **FLEXIO\_I2S\_GetStatusFlags** (**FLEXIO\_I2S\_Type** \*base)  
*Gets the FlexIO I2S status flags.*

## Interrupts

- void **FLEXIO\_I2S\_EnableInterrupts** (**FLEXIO\_I2S\_Type** \*base, uint32\_t mask)  
*Enables the FlexIO I2S interrupt.*
- void **FLEXIO\_I2S\_DisableInterrupts** (**FLEXIO\_I2S\_Type** \*base, uint32\_t mask)  
*Disables the FlexIO I2S interrupt.*

## DMA Control

- static void **FLEXIO\_I2S\_TxEnableDMA** (**FLEXIO\_I2S\_Type** \*base, bool enable)  
*Enables/disables the FlexIO I2S Tx DMA requests.*
- static void **FLEXIO\_I2S\_RxEnableDMA** (**FLEXIO\_I2S\_Type** \*base, bool enable)  
*Enables/disables the FlexIO I2S Rx DMA requests.*
- static uint32\_t **FLEXIO\_I2S\_TxGetDataRegisterAddress** (**FLEXIO\_I2S\_Type** \*base)  
*Gets the FlexIO I2S send data register address.*
- static uint32\_t **FLEXIO\_I2S\_RxGetDataRegisterAddress** (**FLEXIO\_I2S\_Type** \*base)  
*Gets the FlexIO I2S receive data register address.*

## Bus Operations

- void **FLEXIO\_I2S\_MasterSetFormat** (**FLEXIO\_I2S\_Type** \*base, **flexio\_i2s\_format\_t** \*format, uint32\_t srcClock\_Hz)  
*Configures the FlexIO I2S audio format in master mode.*
- void **FLEXIO\_I2S\_SlaveSetFormat** (**FLEXIO\_I2S\_Type** \*base, **flexio\_i2s\_format\_t** \*format)  
*Configures the FlexIO I2S audio format in slave mode.*
- void **FLEXIO\_I2S\_WriteBlocking** (**FLEXIO\_I2S\_Type** \*base, uint8\_t bitWidth, uint8\_t \*txData, size\_t size)  
*Sends a piece of data using a blocking method.*
- static void **FLEXIO\_I2S\_WriteData** (**FLEXIO\_I2S\_Type** \*base, uint8\_t bitWidth, uint32\_t data)

## FlexIO I2S Driver

- Writes a data into data register.  
• void `FLEXIO_I2S_ReadBlocking` (`FLEXIO_I2S_Type` \*base, `uint8_t` bitWidth, `uint8_t` \*rxData, `size_t` size)  
*Receives a piece of data using a blocking method.*
- static `uint32_t FLEXIO_I2S_ReadData` (`FLEXIO_I2S_Type` \*base)  
*Reads a data from the data register.*

## Transactional

- void `FLEXIO_I2S_TransferTxCreateHandle` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_handle_t` \*handle, `flexio_i2s_callback_t` callback, void \*userData)  
*Initializes the FlexIO I2S handle.*
- void `FLEXIO_I2S_TransferSetFormat` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_handle_t` \*handle, `flexio_i2s_format_t` \*format, `uint32_t` srcClock\_Hz)  
*Configures the FlexIO I2S audio format.*
- void `FLEXIO_I2S_TransferRxCreateHandle` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_handle_t` \*handle, `flexio_i2s_callback_t` callback, void \*userData)  
*Initializes the FlexIO I2S receive handle.*
- `status_t FLEXIO_I2S_TransferSendNonBlocking` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_handle_t` \*handle, `flexio_i2s_transfer_t` \*xfer)  
*Performs an interrupt non-blocking send transfer on FlexIO I2S.*
- `status_t FLEXIO_I2S_TransferReceiveNonBlocking` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_handle_t` \*handle, `flexio_i2s_transfer_t` \*xfer)  
*Performs an interrupt non-blocking receive transfer on FlexIO I2S.*
- void `FLEXIO_I2S_TransferAbortSend` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_handle_t` \*handle)  
*Aborts the current send.*
- void `FLEXIO_I2S_TransferAbortReceive` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_handle_t` \*handle)  
*Aborts the current receive.*
- `status_t FLEXIO_I2S_TransferGetSendCount` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_handle_t` \*handle, `size_t` \*count)  
*Gets the remaining bytes to be sent.*
- `status_t FLEXIO_I2S_TransferGetReceiveCount` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_handle_t` \*handle, `size_t` \*count)  
*Gets the remaining bytes to be received.*
- void `FLEXIO_I2S_TransferTxHandleIRQ` (void \*i2sBase, void \*i2sHandle)  
*Tx interrupt handler.*
- void `FLEXIO_I2S_TransferRxHandleIRQ` (void \*i2sBase, void \*i2sHandle)  
*Rx interrupt handler.*

### 15.5.3 Data Structure Documentation

#### 15.5.3.1 struct `FLEXIO_I2S_Type`

##### Data Fields

- `FLEXIO_Type` \* `flexioBase`  
*FlexIO base pointer.*

- `uint8_t txPinIndex`  
*Tx data pin index in FlexIO pins.*
- `uint8_t rxPinIndex`  
*Rx data pin index.*
- `uint8_t bclkPinIndex`  
*Bit clock pin index.*
- `uint8_t fsPinIndex`  
*Frame sync pin index.*
- `uint8_t txShifterIndex`  
*Tx data shifter index.*
- `uint8_t rxShifterIndex`  
*Rx data shifter index.*
- `uint8_t bclkTimerIndex`  
*Bit clock timer index.*
- `uint8_t fsTimerIndex`  
*Frame sync timer index.*

### 15.5.3.2 struct flexio\_i2s\_config\_t

#### Data Fields

- `bool enableI2S`  
*Enable FlexIO I2S.*
- `flexio_i2s_master_slave_t masterSlave`  
*Master or slave.*

### 15.5.3.3 struct flexio\_i2s\_format\_t

#### Data Fields

- `uint8_t bitWidth`  
*Bit width of audio data, always 8/16/24/32 bits.*
- `uint32_t sampleRate_Hz`  
*Sample rate of the audio data.*

### 15.5.3.4 struct flexio\_i2s\_transfer\_t

#### Data Fields

- `uint8_t * data`  
*Data buffer start pointer.*
- `size_t dataSize`  
*Bytes to be transferred.*

## FlexIO I2S Driver

### 15.5.3.4.0.5 Field Documentation

#### 15.5.3.4.0.5.1 size\_t flexio\_i2s\_transfer\_t::dataSize

#### 15.5.3.5 struct \_flexio\_i2s\_handle

##### Data Fields

- `uint32_t state`  
*Internal state.*
- `flexio_i2s_callback_t callback`  
*Callback function called at transfer event.*
- `void *userData`  
*Callback parameter passed to callback function.*
- `uint8_t bitWidth`  
*Bit width for transfer, 8/16/24/32bits.*
- `flexio_i2s_transfer_t queue [FLEXIO_I2S_XFER_QUEUE_SIZE]`  
*Transfer queue storing queued transfer.*
- `size_t transferSize [FLEXIO_I2S_XFER_QUEUE_SIZE]`  
*Data bytes need to transfer.*
- `volatile uint8_t queueUser`  
*Index for user to queue transfer.*
- `volatile uint8_t queueDriver`  
*Index for driver to get the transfer data and size.*

### 15.5.4 Macro Definition Documentation

#### 15.5.4.1 #define FSL\_FLEXIO\_I2S\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))

#### 15.5.4.2 #define FLEXIO\_I2S\_XFER\_QUEUE\_SIZE (4)

### 15.5.5 Enumeration Type Documentation

#### 15.5.5.1 enum \_flexio\_i2s\_status

Enumerator

- `kStatus_FLEXIO_I2S_Idle` FlexIO I2S is in idle state.
- `kStatus_FLEXIO_I2S_TxBusy` FlexIO I2S Tx is busy.
- `kStatus_FLEXIO_I2S_RxBusy` FlexIO I2S Rx is busy.
- `kStatus_FLEXIO_I2S_Error` FlexIO I2S error occurred.
- `kStatus_FLEXIO_I2S_QueueFull` FlexIO I2S transfer queue is full.

### 15.5.5.2 enum flexio\_i2s\_master\_slave\_t

Enumerator

*kFLEXIO\_I2S\_Master* Master mode.

*kFLEXIO\_I2S\_Slave* Slave mode.

### 15.5.5.3 enum \_flexio\_i2s\_interrupt\_enable

Enumerator

*kFLEXIO\_I2S\_TxDataRegEmptyInterruptEnable* Transmit buffer empty interrupt enable.

*kFLEXIO\_I2S\_RxDataRegFullInterruptEnable* Receive buffer full interrupt enable.

### 15.5.5.4 enum \_flexio\_i2s\_status\_flags

Enumerator

*kFLEXIO\_I2S\_TxDataRegEmptyFlag* Transmit buffer empty flag.

*kFLEXIO\_I2S\_RxDataRegFullFlag* Receive buffer full flag.

### 15.5.5.5 enum flexio\_i2s\_sample\_rate\_t

Enumerator

*kFLEXIO\_I2S\_SampleRate8KHz* Sample rate 8000Hz.

*kFLEXIO\_I2S\_SampleRate11025Hz* Sample rate 11025Hz.

*kFLEXIO\_I2S\_SampleRate12KHz* Sample rate 12000Hz.

*kFLEXIO\_I2S\_SampleRate16KHz* Sample rate 16000Hz.

*kFLEXIO\_I2S\_SampleRate22050Hz* Sample rate 22050Hz.

*kFLEXIO\_I2S\_SampleRate24KHz* Sample rate 24000Hz.

*kFLEXIO\_I2S\_SampleRate32KHz* Sample rate 32000Hz.

*kFLEXIO\_I2S\_SampleRate44100Hz* Sample rate 44100Hz.

*kFLEXIO\_I2S\_SampleRate48KHz* Sample rate 48000Hz.

*kFLEXIO\_I2S\_SampleRate96KHz* Sample rate 96000Hz.

### 15.5.5.6 enum flexio\_i2s\_word\_width\_t

Enumerator

*kFLEXIO\_I2S\_WordWidth8bits* Audio data width 8 bits.

*kFLEXIO\_I2S\_WordWidth16bits* Audio data width 16 bits.

*kFLEXIO\_I2S\_WordWidth24bits* Audio data width 24 bits.

*kFLEXIO\_I2S\_WordWidth32bits* Audio data width 32 bits.

## FlexIO I2S Driver

### 15.5.6 Function Documentation

#### 15.5.6.1 void FLEXIO\_I2S\_Init ( FLEXIO\_I2S\_Type \* *base*, const flexio\_i2s\_config\_t \* *config* )

This API configures FlexIO pins and shifter to I2S and configure FlexIO I2S with configuration structure. The configuration structure can be filled by the user, or be set with default values by [FLEXIO\\_I2S\\_GetDefaultConfig\(\)](#).

#### Note

This API should be called at the beginning of the application to use the FlexIO I2S driver, or any access to the FlexIO I2S module could cause hard fault because clock is not enabled.

#### Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | FlexIO I2S base pointer         |
| <i>config</i> | FlexIO I2S configure structure. |

#### 15.5.6.2 void FLEXIO\_I2S\_GetDefaultConfig ( flexio\_i2s\_config\_t \* *config* )

The purpose of this API is to get the configuration structure initialized for use in [FLEXIO\\_I2S\\_Init\(\)](#). User may use the initialized structure unchanged in [FLEXIO\\_I2S\\_Init\(\)](#), or modify some fields of the structure before calling [FLEXIO\\_I2S\\_Init\(\)](#).

#### Parameters

|               |                                           |
|---------------|-------------------------------------------|
| <i>config</i> | pointer to master configuration structure |
|---------------|-------------------------------------------|

#### 15.5.6.3 void FLEXIO\_I2S\_Deinit ( FLEXIO\_I2S\_Type \* *base* )

Calling this API gates the FlexIO i2s clock. After calling this API, call the [FLEXIO\\_I2S\\_Init](#) to use the FlexIO I2S module.

#### Parameters

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

#### 15.5.6.4 static void FLEXIO\_I2S\_Enable ( FLEXIO\_I2S\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_I2S_Type</a> |
| <i>enable</i> | True to enable, false to disable.          |

#### 15.5.6.5 `uint32_t FLEXIO_I2S_GetStatusFlags ( FLEXIO_I2S_Type * base )`

Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
|-------------|------------------------------------------------------|

Returns

Status flag, which are ORed by the enumerators in the \_flexio\_i2s\_status\_flags.

#### 15.5.6.6 `void FLEXIO_I2S_EnableInterrupts ( FLEXIO_I2S_Type * base, uint32_t mask )`

This function enables the FlexIO UART interrupt.

Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
| <i>mask</i> | interrupt source                                     |

#### 15.5.6.7 `void FLEXIO_I2S_DisableInterrupts ( FLEXIO_I2S_Type * base, uint32_t mask )`

This function disables the FlexIO UART interrupt.

Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
| <i>mask</i> | interrupt source                                     |

#### 15.5.6.8 `static void FLEXIO_I2S_TxEnableDMA ( FLEXIO_I2S_Type * base, bool enable ) [inline], [static]`

## FlexIO I2S Driver

Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>base</i>   | FlexIO I2S base pointer                         |
| <i>enable</i> | True means enable DMA, false means disable DMA. |

**15.5.6.9 static void FLEXIO\_I2S\_RxEnableDMA ( FLEXIO\_I2S\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>base</i>   | FlexIO I2S base pointer                         |
| <i>enable</i> | True means enable DMA, false means disable DMA. |

**15.5.6.10 static uint32\_t FLEXIO\_I2S\_TxGetDataRegisterAddress ( FLEXIO\_I2S\_Type \* *base* ) [inline], [static]**

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

|             |                                             |
|-------------|---------------------------------------------|
| <i>base</i> | pointer to <b>FLEXIO_I2S_Type</b> structure |
|-------------|---------------------------------------------|

Returns

FlexIO i2s send data register address.

**15.5.6.11 static uint32\_t FLEXIO\_I2S\_RxGetDataRegisterAddress ( FLEXIO\_I2S\_Type \* *base* ) [inline], [static]**

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

|             |                                             |
|-------------|---------------------------------------------|
| <i>base</i> | pointer to <b>FLEXIO_I2S_Type</b> structure |
|-------------|---------------------------------------------|

Returns

FlexIO i2s receive data register address.

**15.5.6.12 void FLEXIO\_I2S\_MasterSetFormat ( FLEXIO\_I2S\_Type \* *base*,  
flexio\_i2s\_format\_t \* *format*, uint32\_t *srcClock\_Hz* )**

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

## FlexIO I2S Driver

Parameters

|                    |                                                      |
|--------------------|------------------------------------------------------|
| <i>base</i>        | pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
| <i>format</i>      | Pointer to FlexIO I2S audio data format structure.   |
| <i>srcClock_Hz</i> | I2S master clock source frequency in Hz.             |

### 15.5.6.13 void [FLEXIO\\_I2S\\_SlaveSetFormat](#) ( [FLEXIO\\_I2S\\_Type](#) \* *base*, [flexio\\_i2s\\_format\\_t](#) \* *format* )

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
| <i>format</i> | Pointer to FlexIO I2S audio data format structure.   |

### 15.5.6.14 void [FLEXIO\\_I2S\\_WriteBlocking](#) ( [FLEXIO\\_I2S\\_Type](#) \* *base*, [uint8\\_t](#) *bitWidth*, [uint8\\_t](#) \* *txData*, [size\\_t](#) *size* )

Note

This function blocks via polling until data is ready to be sent.

Parameters

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>base</i>     | FlexIO I2S base pointer.                                |
| <i>bitWidth</i> | How many bits in a audio word, usually 8/16/24/32 bits. |
| <i>txData</i>   | Pointer to the data to be written.                      |
| <i>size</i>     | Bytes to be written.                                    |

### 15.5.6.15 static void [FLEXIO\\_I2S\\_WriteData](#) ( [FLEXIO\\_I2S\\_Type](#) \* *base*, [uint8\\_t](#) *bitWidth*, [uint32\\_t](#) *data* ) [inline], [static]

Parameters

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>base</i>     | FlexIO I2S base pointer.                                |
| <i>bitWidth</i> | How many bits in a audio word, usually 8/16/24/32 bits. |
| <i>data</i>     | Data to be written.                                     |

**15.5.6.16 void FLEXIO\_I2S\_ReadBlocking ( FLEXIO\_I2S\_Type \* *base*, uint8\_t *bitWidth*, uint8\_t \* *rxData*, size\_t *size* )**

Note

This function blocks via polling until data is ready to be sent.

Parameters

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>base</i>     | FlexIO I2S base pointer                                 |
| <i>bitWidth</i> | How many bits in a audio word, usually 8/16/24/32 bits. |
| <i>rxData</i>   | Pointer to the data to be read.                         |
| <i>size</i>     | Bytes to be read.                                       |

**15.5.6.17 static uint32\_t FLEXIO\_I2S\_ReadData ( FLEXIO\_I2S\_Type \* *base* )  
[inline], [static]**

Parameters

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

Returns

Data read from data register.

**15.5.6.18 void FLEXIO\_I2S\_TransferTxCreateHandle ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle*, flexio\_i2s\_callback\_t *callback*, void \* *userData* )**

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

## FlexIO I2S Driver

Parameters

|                 |                                                                                       |
|-----------------|---------------------------------------------------------------------------------------|
| <i>base</i>     | pointer to <a href="#">FLEXIO_I2S_Type</a> structure                                  |
| <i>handle</i>   | pointer to <a href="#">flexio_i2s_handle_t</a> structure to store the transfer state. |
| <i>callback</i> | FlexIO I2S callback function, which is called while finished a block.                 |
| <i>userData</i> | User parameter for the FlexIO I2S callback.                                           |

**15.5.6.19 void FLEXIO\_I2S\_TransferSetFormat ( [FLEXIO\\_I2S\\_Type](#) \* *base*, [flexio\\_i2s\\_handle\\_t](#) \* *handle*, [flexio\\_i2s\\_format\\_t](#) \* *format*, [uint32\\_t](#) *srcClock\_Hz* )**

Audio format can be changed in run-time of FlexIO i2s. This function configures the sample rate and audio data format to be transferred.

Parameters

|                    |                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------|
| <i>base</i>        | pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                                        |
| <i>handle</i>      | FlexIO I2S handle pointer.                                                                   |
| <i>format</i>      | Pointer to audio data format structure.                                                      |
| <i>srcClock_Hz</i> | FlexIO I2S bit clock source frequency in Hz. This parameter should be 0 while in slave mode. |

**15.5.6.20 void FLEXIO\_I2S\_TransferRxCreateHandle ( [FLEXIO\\_I2S\\_Type](#) \* *base*, [flexio\\_i2s\\_handle\\_t](#) \* *handle*, [flexio\\_i2s\\_callback\\_t](#) *callback*, [void](#) \* *userData* )**

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Usually, user only need to call this API once to get the initialized handle.

Parameters

|                 |                                                                                       |
|-----------------|---------------------------------------------------------------------------------------|
| <i>base</i>     | pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                                 |
| <i>handle</i>   | pointer to <a href="#">flexio_i2s_handle_t</a> structure to store the transfer state. |
| <i>callback</i> | FlexIO I2S callback function, which is called while finished a block.                 |
| <i>userData</i> | User parameter for the FlexIO I2S callback.                                           |

**15.5.6.21 [status\\_t](#) FLEXIO\_I2S\_TransferSendNonBlocking ( [FLEXIO\\_I2S\\_Type](#) \* *base*, [flexio\\_i2s\\_handle\\_t](#) \* *handle*, [flexio\\_i2s\\_transfer\\_t](#) \* *xfer* )**

## Note

Calling the API returns immediately after transfer initiates. Call FLEXIO\_I2S\_GetRemainingBytes to poll the transfer status and check whether the transfer is finished. If the return status is 0, the transfer is finished.

## Parameters

|               |                                                                                          |
|---------------|------------------------------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                                    |
| <i>handle</i> | pointer to <a href="#">flexio_i2s_handle_t</a> structure which stores the transfer state |
| <i>xfer</i>   | pointer to <a href="#">flexio_i2s_transfer_t</a> structure                               |

## Return values

|                                   |                                                                                    |
|-----------------------------------|------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>            | Successfully start the data transmission.                                          |
| <i>kStatus_FLEXIO_I2S_Tx-Busy</i> | Previous transmission still not finished, data not all written to TX register yet. |
| <i>kStatus_InvalidArgument</i>    | The input parameter is invalid.                                                    |

### 15.5.6.22 **status\_t FLEXIO\_I2S\_TransferReceiveNonBlocking ( [FLEXIO\\_I2S\\_Type](#) \* *base*, [flexio\\_i2s\\_handle\\_t](#) \* *handle*, [flexio\\_i2s\\_transfer\\_t](#) \* *xfer* )**

## Note

The API returns immediately after transfer initiates. Call FLEXIO\_I2S\_GetRemainingBytes to poll the transfer status to check whether the transfer is finished. If the return status is 0, the transfer is finished.

## Parameters

|               |                                                                                          |
|---------------|------------------------------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                                    |
| <i>handle</i> | pointer to <a href="#">flexio_i2s_handle_t</a> structure which stores the transfer state |
| <i>xfer</i>   | pointer to <a href="#">flexio_i2s_transfer_t</a> structure                               |

## Return values

|                                  |                                      |
|----------------------------------|--------------------------------------|
| <i>kStatus_Success</i>           | Successfully start the data receive. |
| <i>kStatus_FLEXIO_I2S-RxBusy</i> | Previous receive still not finished. |
| <i>kStatus_InvalidArgument</i>   | The input parameter is invalid.      |

## FlexIO I2S Driver

**15.5.6.23 void FLEXIO\_I2S\_TransferAbortSend ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle* )**

Note

This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                    |
| <i>handle</i> | pointer to flexio_i2s_handle_t structure which stores the transfer state |

**15.5.6.24 void FLEXIO\_I2S\_TransferAbortReceive ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle* )**

Note

This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                    |
| <i>handle</i> | pointer to flexio_i2s_handle_t structure which stores the transfer state |

**15.5.6.25 status\_t FLEXIO\_I2S\_TransferGetSendCount ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                    |
| <i>handle</i> | pointer to flexio_i2s_handle_t structure which stores the transfer state |
| <i>count</i>  | Bytes sent.                                                              |

Return values

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

15.5.6.26 **status\_t FLEXIO\_I2S\_TransferGetReceiveCount( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle*, size\_t \* *count* )**

## FlexIO I2S Driver

Parameters

|               |                                                                                       |
|---------------|---------------------------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                                 |
| <i>handle</i> | pointer to <code>flexio_i2s_handle_t</code> structure which stores the transfer state |

Returns

count Bytes received.

Return values

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

### 15.5.6.27 void FLEXIO\_I2S\_TransferTxHandleIRQ ( `void * i2sBase, void * i2sHandle` )

Parameters

|                  |                                                       |
|------------------|-------------------------------------------------------|
| <i>i2sBase</i>   | pointer to <a href="#">FLEXIO_I2S_Type</a> structure. |
| <i>i2sHandle</i> | pointer to <code>flexio_i2s_handle_t</code> structure |

### 15.5.6.28 void FLEXIO\_I2S\_TransferRxHandleIRQ ( `void * i2sBase, void * i2sHandle` )

Parameters

|                  |                                                       |
|------------------|-------------------------------------------------------|
| <i>i2sBase</i>   | pointer to <a href="#">FLEXIO_I2S_Type</a> structure. |
| <i>i2sHandle</i> | pointer to <code>flexio_i2s_handle_t</code> structure |

## 15.5.7 FlexIO eDMA I2S Driver

### 15.5.7.1 Overview

#### Data Structures

- struct `flexio_i2s_edma_handle_t`

*FlexIO I2S DMA transfer handle, users should not touch the content of the handle.* [More...](#)

#### TypeDefs

- typedef void(\* `flexio_i2s_edma_callback_t`)(`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle, `status_t` status, `void` \*userData)

*FlexIO I2S eDMA transfer callback function for finish and error.*

#### eDMA Transactional

- void `FLEXIO_I2S_TransferTxCreateHandleEDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle, `flexio_i2s_edma_callback_t` callback, `void` \*userData, `edma_handle_t` \*dmaHandle)

*Initializes the FlexIO I2S eDMA handle.*

- void `FLEXIO_I2S_TransferRxCreateHandleEDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle, `flexio_i2s_edma_callback_t` callback, `void` \*userData, `edma_handle_t` \*dmaHandle)

*Initializes the FlexIO I2S Rx eDMA handle.*

- void `FLEXIO_I2S_TransferSetFormatEDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle, `flexio_i2s_format_t` \*format, `uint32_t` srcClock\_Hz)

*Configures the FlexIO I2S Tx audio format.*

- `status_t FLEXIO_I2S_TransferSendEDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle, `flexio_i2s_transfer_t` \*xfer)

*Performs a non-blocking FlexIO I2S transfer using DMA.*

- `status_t FLEXIO_I2S_TransferReceiveEDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle, `flexio_i2s_transfer_t` \*xfer)

*Performs a non-blocking FlexIO I2S receive using eDMA.*

- void `FLEXIO_I2S_TransferAbortSendEDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle)

*Aborts a FlexIO I2S transfer using eDMA.*

- void `FLEXIO_I2S_TransferAbortReceiveEDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle)

*Aborts a FlexIO I2S receive using eDMA.*

- `status_t FLEXIO_I2S_TransferGetSendCountEDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle, `size_t` \*count)

*Gets the remaining bytes to be sent.*

- `status_t FLEXIO_I2S_TransferGetReceiveCountEDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle, `size_t` \*count)

*Get the remaining bytes to be received.*

## FlexIO I2S Driver

### 15.5.7.2 Data Structure Documentation

#### 15.5.7.2.1 struct \_flexio\_i2s\_edma\_handle

##### Data Fields

- **edma\_handle\_t \* dmaHandle**  
*DMA handler for FlexIO I2S send.*
- **uint8\_t bytesPerFrame**  
*Bytes in a frame.*
- **uint32\_t state**  
*Internal state for FlexIO I2S eDMA transfer.*
- **flexio\_i2s\_edma\_callback\_t callback**  
*Callback for users while transfer finish or error occurred.*
- **void \* userData**  
*User callback parameter.*
- **edma\_tcd\_t tcd [FLEXIO\_I2S\_XFER\_QUEUE\_SIZE+1U]**  
*TCD pool for eDMA transfer.*
- **flexio\_i2s\_transfer\_t queue [FLEXIO\_I2S\_XFER\_QUEUE\_SIZE]**  
*Transfer queue storing queued transfer.*
- **size\_t transferSize [FLEXIO\_I2S\_XFER\_QUEUE\_SIZE]**  
*Data bytes need to transfer.*
- **volatile uint8\_t queueUser**  
*Index for user to queue transfer.*
- **volatile uint8\_t queueDriver**  
*Index for driver to get the transfer data and size.*

#### 15.5.7.2.1.1 Field Documentation

15.5.7.2.1.1.1 **edma\_tcd\_t flexio\_i2s\_edma\_handle\_t::tcd[FLEXIO\_I2S\_XFER\_QUEUE\_SIZE+1U]**

15.5.7.2.1.1.2 **flexio\_i2s\_transfer\_t flexio\_i2s\_edma\_handle\_t::queue[FLEXIO\_I2S\_XFER\_QUEUE\_SIZE]**

15.5.7.2.1.1.3 **volatile uint8\_t flexio\_i2s\_edma\_handle\_t::queueUser**

#### 15.5.7.3 Function Documentation

15.5.7.3.1 **void FLEXIO\_I2S\_TransferTxCreateHandleEDMA ( FLEXIO\_I2S\_Type \* base, flexio\_i2s\_edma\_handle\_t \* handle, flexio\_i2s\_edma\_callback\_t callback, void \* userData, edma\_handle\_t \* dmaHandle )**

This function initializes the FlexIO I2S master DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, user need only call this API once to get the initialized handle.

Parameters

|                  |                                                                                     |
|------------------|-------------------------------------------------------------------------------------|
| <i>base</i>      | FlexIO I2S peripheral base address.                                                 |
| <i>handle</i>    | FlexIO I2S eDMA handle pointer.                                                     |
| <i>callback</i>  | FlexIO I2S eDMA callback function called while finished a block.                    |
| <i>userData</i>  | User parameter for callback.                                                        |
| <i>dmaHandle</i> | eDMA handle for FlexIO I2S. This handle shall be a static value allocated by users. |

**15.5.7.3.2 void FLEXIO\_I2S\_TransferRxCreateHandleEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle*, flexio\_i2s\_edma\_callback\_t *callback*, void \* *userData*, edma\_handle\_t \* *dmaHandle* )**

This function initializes the FlexIO I2S slave DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, user need only call this API once to get the initialized handle.

Parameters

|                  |                                                                                     |
|------------------|-------------------------------------------------------------------------------------|
| <i>base</i>      | FlexIO I2S peripheral base address.                                                 |
| <i>handle</i>    | FlexIO I2S eDMA handle pointer.                                                     |
| <i>callback</i>  | FlexIO I2S eDMA callback function called while finished a block.                    |
| <i>userData</i>  | User parameter for callback.                                                        |
| <i>dmaHandle</i> | eDMA handle for FlexIO I2S. This handle shall be a static value allocated by users. |

**15.5.7.3.3 void FLEXIO\_I2S\_TransferSetFormatEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle*, flexio\_i2s\_format\_t \* *format*, uint32\_t *srcClock\_Hz* )**

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred. This function also sets eDMA parameter according to format.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S eDMA handle pointer      |

## FlexIO I2S Driver

|                    |                                                                              |
|--------------------|------------------------------------------------------------------------------|
| <i>format</i>      | Pointer to FlexIO I2S audio data format structure.                           |
| <i>srcClock_Hz</i> | FlexIO I2S clock source frequency in Hz, it should be 0 while in slave mode. |

Return values

|                                |                                 |
|--------------------------------|---------------------------------|
| <i>kStatus_Success</i>         | Audio format set successfully.  |
| <i>kStatus_InvalidArgument</i> | The input arguments is invalid. |

**15.5.7.3.4 status\_t FLEXIO\_I2S\_TransferSendEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle*, flexio\_i2s\_transfer\_t \* *xfer* )**

Note

This interface returned immediately after transfer initiates, users should call FLEXIO\_I2S\_GetTransferStatus to poll the transfer status to check whether FlexIO I2S transfer finished.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |
| <i>xfer</i>   | Pointer to DMA transfer structure.  |

Return values

|                                |                                            |
|--------------------------------|--------------------------------------------|
| <i>kStatus_Success</i>         | Start a FlexIO I2S eDMA send successfully. |
| <i>kStatus_InvalidArgument</i> | The input arguments is invalid.            |
| <i>kStatus_TxBusy</i>          | FlexIO I2S is busy sending data.           |

**15.5.7.3.5 status\_t FLEXIO\_I2S\_TransferReceiveEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle*, flexio\_i2s\_transfer\_t \* *xfer* )**

Note

This interface returned immediately after transfer initiates, users should call FLEXIO\_I2S\_GetReceiveRemainingBytes to poll the transfer status to check whether FlexIO I2S transfer finished.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |
| <i>xfer</i>   | Pointer to DMA transfer structure.  |

Return values

|                                |                                               |
|--------------------------------|-----------------------------------------------|
| <i>kStatus_Success</i>         | Start a FlexIO I2S eDMA receive successfully. |
| <i>kStatus_InvalidArgument</i> | The input arguments is invalid.               |
| <i>kStatus_RxBusy</i>          | FlexIO I2S is busy receiving data.            |

#### 15.5.7.3.6 void FLEXIO\_I2S\_TransferAbortSendEDMA ( **FLEXIO\_I2S\_Type \* base,** **flexio\_i2s\_edma\_handle\_t \* handle** )

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |

#### 15.5.7.3.7 void FLEXIO\_I2S\_TransferAbortReceiveEDMA ( **FLEXIO\_I2S\_Type \* base,** **flexio\_i2s\_edma\_handle\_t \* handle** )

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |

#### 15.5.7.3.8 status\_t FLEXIO\_I2S\_TransferGetSendCountEDMA ( **FLEXIO\_I2S\_Type \* base,** **flexio\_i2s\_edma\_handle\_t \* handle, size\_t \* count** )

Parameters

|             |                                     |
|-------------|-------------------------------------|
| <i>base</i> | FlexIO I2S peripheral base address. |
|-------------|-------------------------------------|

## FlexIO I2S Driver

|               |                                |
|---------------|--------------------------------|
| <i>handle</i> | FlexIO I2S DMA handle pointer. |
| <i>count</i>  | Bytes sent.                    |

Return values

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

**15.5.7.3.9 status\_t FLEXIO\_I2S\_TransferGetReceiveCountEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |
| <i>count</i>  | Bytes received.                     |

Return values

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

## 15.5.8 FlexIO DMA I2S Driver

### 15.5.8.1 Overview

#### Data Structures

- struct `flexio_i2s_dma_handle_t`

*FlexIO I2S DMA transfer handle, users should not touch the content of the handle.* [More...](#)

#### TypeDefs

- typedef void(\* `flexio_i2s_dma_callback_t`)(`FLEXIO_I2S_Type` \*base, `flexio_i2s_dma_handle_t` \*handle, `status_t` status, `void` \*userData)

*FlexIO I2S DMA transfer callback function for finish and error.*

#### DMA Transactional

- void `FLEXIO_I2S_TransferTxCreateHandleDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_dma_handle_t` \*handle, `flexio_i2s_dma_callback_t` callback, `void` \*userData, `dma_handle_t` \*dmaHandle)
- Initializes the FlexIO I2S DMA handle.*
- void `FLEXIO_I2S_TransferRxCreateHandleDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_dma_handle_t` \*handle, `flexio_i2s_dma_callback_t` callback, `void` \*userData, `dma_handle_t` \*dmaHandle)
- Initializes the FlexIO I2S Rx DMA handle.*
- void `FLEXIO_I2S_TransferSetFormatDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_dma_handle_t` \*handle, `flexio_i2s_format_t` \*format, `uint32_t` srcClock\_Hz)
- Configures the FlexIO I2S Tx audio format.*
- `status_t FLEXIO_I2S_TransferSendDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_dma_handle_t` \*handle, `flexio_i2s_transfer_t` \*xfer)
- Performs a non-blocking FlexIO I2S transfer using DMA.*
- `status_t FLEXIO_I2S_TransferReceiveDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_dma_handle_t` \*handle, `flexio_i2s_transfer_t` \*xfer)
- Performs a non-blocking FlexIO I2S receive using DMA.*
- void `FLEXIO_I2S_TransferAbortSendDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_dma_handle_t` \*handle)
- Aborts a FlexIO I2S transfer using DMA.*
- void `FLEXIO_I2S_TransferAbortReceiveDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_dma_handle_t` \*handle)
- Aborts a FlexIO I2S receive using DMA.*
- `status_t FLEXIO_I2S_TransferGetSendCountDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_dma_handle_t` \*handle, `size_t` \*count)
- Gets the remaining bytes to be sent.*
- `status_t FLEXIO_I2S_TransferGetReceiveCountDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_dma_handle_t` \*handle, `size_t` \*count)
- Gets the remaining bytes to be received.*

## FlexIO I2S Driver

### 15.5.8.2 Data Structure Documentation

#### 15.5.8.2.1 struct \_flexio\_i2s\_dma\_handle

##### Data Fields

- `dma_handle_t * dmaHandle`  
*DMA handler for FlexIO I2S send.*
- `uint8_t bytesPerFrame`  
*Bytes in a frame.*
- `uint32_t state`  
*Internal state for FlexIO I2S DMA transfer.*
- `flexio_i2s_dma_callback_t callback`  
*Callback for users while transfer finish or error occurred.*
- `void * userData`  
*User callback parameter.*
- `flexio_i2s_transfer_t queue [FLEXIO_I2S_XFER_QUEUE_SIZE]`  
*Transfer queue storing queued transfer.*
- `size_t transferSize [FLEXIO_I2S_XFER_QUEUE_SIZE]`  
*Data bytes need to transfer.*
- `volatile uint8_t queueUser`  
*Index for user to queue transfer.*
- `volatile uint8_t queueDriver`  
*Index for driver to get the transfer data and size.*

#### 15.5.8.2.1.1 Field Documentation

15.5.8.2.1.1.1 `flexio_i2s_transfer_t flexio_i2s_dma_handle_t::queue[FLEXIO_I2S_XFER_QUEUE_SIZE]`

15.5.8.2.1.1.2 `volatile uint8_t flexio_i2s_dma_handle_t::queueUser`

### 15.5.8.3 Function Documentation

15.5.8.3.1 `void FLEXIO_I2S_TransferTxCreateHandleDMA ( FLEXIO_I2S_Type * base,  
flexio_i2s_dma_handle_t * handle, flexio_i2s_dma_callback_t callback, void *  
userData, dma_handle_t * dmaHandle )`

This function initializes the FlexIO I2S master DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, user need only call this API once to get the initialized handle.

Parameters

|                  |                                                                                    |
|------------------|------------------------------------------------------------------------------------|
| <i>base</i>      | FlexIO I2S peripheral base address.                                                |
| <i>handle</i>    | FlexIO I2S DMA handle pointer.                                                     |
| <i>callback</i>  | FlexIO I2S DMA callback function called while finished a block.                    |
| <i>userData</i>  | User parameter for callback.                                                       |
| <i>dmaHandle</i> | DMA handle for FlexIO I2S. This handle shall be a static value allocated by users. |

**15.5.8.3.2 void FLEXIO\_I2S\_TransferRxCreateHandleDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_dma\_handle\_t \* *handle*, flexio\_i2s\_dma\_callback\_t *callback*, void \* *userData*, dma\_handle\_t \* *dmaHandle* )**

This function initializes the FlexIO I2S slave DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, user need only call this API once to get the initialized handle.

Parameters

|                  |                                                                                    |
|------------------|------------------------------------------------------------------------------------|
| <i>base</i>      | FlexIO I2S peripheral base address.                                                |
| <i>handle</i>    | FlexIO I2S DMA handle pointer.                                                     |
| <i>callback</i>  | FlexIO I2S DMA callback function called while finished a block.                    |
| <i>userData</i>  | User parameter for callback.                                                       |
| <i>dmaHandle</i> | DMA handle for FlexIO I2S. This handle shall be a static value allocated by users. |

**15.5.8.3.3 void FLEXIO\_I2S\_TransferSetFormatDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_dma\_handle\_t \* *handle*, flexio\_i2s\_format\_t \* *format*, uint32\_t *srcClock\_Hz* )**

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred. This function also sets DMA parameter according to format.

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address.                |
| <i>handle</i> | FlexIO I2S DMA handle pointer                      |
| <i>format</i> | Pointer to FlexIO I2S audio data format structure. |

## FlexIO I2S Driver

|                    |                                                                              |
|--------------------|------------------------------------------------------------------------------|
| <i>srcClock_Hz</i> | FlexIO I2S clock source frequency in Hz. It should be 0 while in slave mode. |
|--------------------|------------------------------------------------------------------------------|

Return values

|                                |                                 |
|--------------------------------|---------------------------------|
| <i>kStatus_Success</i>         | Audio format set successfully.  |
| <i>kStatus_InvalidArgument</i> | The input arguments is invalid. |

### 15.5.8.3.4 status\_t FLEXIO\_I2S\_TransferSendDMA ( **FLEXIO\_I2S\_Type** \* *base*, **flexio\_i2s\_dma\_handle\_t** \* *handle*, **flexio\_i2s\_transfer\_t** \* *xfer* )

Note

This interface returns immediately after transfer initiates. Call FLEXIO\_I2S\_GetTransferStatus to poll the transfer status and check whether FLEXIO I2S transfer finished.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |
| <i>xfer</i>   | Pointer to DMA transfer structure.  |

Return values

|                                |                                           |
|--------------------------------|-------------------------------------------|
| <i>kStatus_Success</i>         | Start a FlexIO I2S DMA send successfully. |
| <i>kStatus_InvalidArgument</i> | The input arguments is invalid.           |
| <i>kStatus_TxBusy</i>          | FlexIO I2S is busy sending data.          |

### 15.5.8.3.5 status\_t FLEXIO\_I2S\_TransferReceiveDMA ( **FLEXIO\_I2S\_Type** \* *base*, **flexio\_i2s\_dma\_handle\_t** \* *handle*, **flexio\_i2s\_transfer\_t** \* *xfer* )

Note

This interface returns immediately after transfer initiates. Call FLEXIO\_I2S\_GetReceiveRemainingBytes to poll the transfer status to check whether the FlexIO I2S transfer is finished.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |
| <i>xfer</i>   | Pointer to DMA transfer structure.  |

Return values

|                                |                                              |
|--------------------------------|----------------------------------------------|
| <i>kStatus_Success</i>         | Start a FlexIO I2S DMA receive successfully. |
| <i>kStatus_InvalidArgument</i> | The input arguments is invalid.              |
| <i>kStatus_RxBusy</i>          | FlexIO I2S is busy receiving data.           |

#### 15.5.8.3.6 void FLEXIO\_I2S\_TransferAbortSendDMA ( **FLEXIO\_I2S\_Type \* base,**                  **flexio\_i2s\_dma\_handle\_t \* handle** )

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |

#### 15.5.8.3.7 void FLEXIO\_I2S\_TransferAbortReceiveDMA ( **FLEXIO\_I2S\_Type \* base,**                  **flexio\_i2s\_dma\_handle\_t \* handle** )

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |

#### 15.5.8.3.8 status\_t FLEXIO\_I2S\_TransferGetSendCountDMA ( **FLEXIO\_I2S\_Type \* base,**                  **flexio\_i2s\_dma\_handle\_t \* handle,** **size\_t \* count** )

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |

## FlexIO I2S Driver

|              |             |
|--------------|-------------|
| <i>count</i> | Bytes sent. |
|--------------|-------------|

Return values

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

### 15.5.8.3.9 status\_t FLEXIO\_I2S\_TransferGetReceiveCountDMA ( **FLEXIO\_I2S\_Type \* base,** **flexio\_i2s\_dma\_handle\_t \* handle, size\_t \* count** )

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |
| <i>count</i>  | Bytes received.                     |

Return values

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

## 15.6 FlexIO SPI Driver

### 15.6.1 Overview

The KSDK provides a peripheral driver for an SPI function using the Flexible I/O module of Kinetis devices.

FlexIO SPI driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for FlexIO SPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FlexIO SPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the [FLEXIO\\_SPI\\_Type \\*base](#) as the first parameter. FlexIO SPI functional operation groups provide the functional API set.

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs can satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the [flexio\\_spi\\_master\\_handle\\_t/flexio\\_spi\\_slave\\_handle\\_t](#) as the second parameter. Initialize the handle by calling the [FLEXIO\\_SPI\\_MasterTransferCreateHandle\(\)](#) or [FLEXIO\\_SPI\\_SlaveTransferCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [FLEXIO\\_SPI\\_MasterTransferNonBlocking\(\)](#)/[FLEXIO\\_SPI\\_SlaveTransferNonBlocking\(\)](#) set up an interrupt for data transfer. When the transfer is complete, the upper layer is notified through a callback function with the [kStatus\\_FLEXIO\\_SPI\\_Idle](#) status. Notice that FlexIO SPI slave driver only supports discontinuous PCS access, which is a limitation. The FlexIO SPI slave driver could support continuous PCS, but the slave can't adapt discontinuous and continuous PCS automatically. User can change the timer disable mode in [FLEXIO\\_SPI\\_SlaveInit](#) manually, from [kFLEXIO\\_TimerDisableOnTimerCompare](#) to [kFLEXIO\\_TimerDisableNever](#) to enable discontinuous PCS access. Only CPHA = 0 is supported.

### 15.6.2 Typical use case

#### 15.6.2.1 FlexIO SPI send/receive using an interrupt method

```
flexio_spi_master_handle_t g_spiHandle;
FLEXIO_SPI_Type spiDev;
volatile bool txFinished;
static uint8_t srcBuff[BUFFER_SIZE];
static uint8_t destBuff[BUFFER_SIZE];

void FLEXIO_SPI_MasterUserCallback(FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle
    , status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_SPI_Idle == status)
    {
        txFinished = true;
    }
}

void main(void)
```

## FlexIO SPI Driver

```
{  
    //...  
    flexio_spi_transfer_t xfer = {0};  
    flexio_spi_master_config_t userConfig;  
  
    FLEXIO_SPI_MasterGetDefaultConfig(&userConfig);  
    userConfig.baudRate_Bps = 500000U;  
  
    spiDev.flexioBase = BOARD_FLEXIO_BASE;  
    spiDev.SDOPinIndex = FLEXIO_SPI_MOSI_PIN;  
    spiDev.SDIPinIndex = FLEXIO_SPI_MISO_PIN;  
    spiDev.SCKPinIndex = FLEXIO_SPI_SCK_PIN;  
    spiDev.CSnPinIndex = FLEXIO_SPI_CSn_PIN;  
    spiDev.shifterIndex[0] = 0U;  
    spiDev.shifterIndex[1] = 1U;  
    spiDev.timerIndex[0] = 0U;  
    spiDev.timerIndex[1] = 1U;  
  
    FLEXIO_SPI_MasterInit(&spiDev, &userConfig, FLEXIO_CLOCK_FREQUENCY);  
  
    xfer.txData = srcBuff;  
    xfer.rxData = destBuff;  
    xfer.dataSize = BUFFER_SIZE;  
    xfer.flags = kFLEXIO_SPI_8bitMsb;  
    FLEXIO_SPI_MasterTransferCreateHandle(&spiDev, &g_spiHandle,  
        FLEXIO_SPI_MasterUserCallback, NULL);  
    FLEXIO_SPI_MasterTransferNonBlocking(&spiDev, &g_spiHandle, &xfer);  
  
    // Send finished.  
    while (!txFinished)  
    {  
    }  
  
    // ...  
}
```

### 15.6.2.2 FlexIO\_SPI Send/Receive using a DMA method

```
dma_handle_t g_spiTxDmaHandle;  
dma_handle_t g_spiRxDmaHandle;  
flexio_spi_master_handle_t g_spiHandle;  
FLEXIO_SPI_Type spiDev;  
volatile bool txFinished;  
static uint8_t srcBuff[BUFFER_SIZE];  
static uint8_t destBuff[BUFFER_SIZE];  
void FLEXIO_SPI_MasterUserCallback(FLEXIO_SPI_Type *base, flexio_spi_master_dma_handle_t *  
    handle, status_t status, void *userData)  
{  
    userData = userData;  
  
    if (kStatus_FLEXIO_SPI_Idle == status)  
    {  
        txFinished = true;  
    }  
}  
  
void main(void)  
{  
    flexio_spi_transfer_t xfer = {0};  
    flexio_spi_master_config_t userConfig;  
  
    FLEXIO_SPI_MasterGetDefaultConfig(&userConfig);  
    userConfig.baudRate_Bps = 500000U;  
  
    spiDev.flexioBase = BOARD_FLEXIO_BASE;
```

```

spiDev.SDOPinIndex = FLEXIO_SPI_MOSI_PIN;
spiDev.SDIPinIndex = FLEXIO_SPI_MISO_PIN;
spiDev.SCKPinIndex = FLEXIO_SPI_SCK_PIN;
spiDev.CSnPinIndex = FLEXIO_SPI_CSn_PIN;
spiDev.shifterIndex[0] = 0U;
spiDev.shifterIndex[1] = 1U;
spiDev.timerIndex[0] = 0U;
spiDev.timerIndex[1] = 1U;

/*Initializes the DMA for the example.*/
DMAMGR_Init();

dma_request_source_tx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + spiDev.
    shifterIndex[0]);
dma_request_source_rx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + spiDev.
    shifterIndex[1]);

/* Requests DMA channels for transmit and receive. */
DMAMGR_RequestChannel((dma_request_source_t)dma_request_source_tx, 0, &txHandle);
DMAMGR_RequestChannel((dma_request_source_t)dma_request_source_rx, 1, &rxHandle);

FLEXIO_SPI_MasterInit(&spiDev, &userConfig, FLEXIO_CLOCK_FREQUENCY);

/* Initializes the buffer. */
for (i = 0; i < BUFFER_SIZE; i++)
{
    srcBuff[i] = i;
}

/* Sends to the slave. */
xfer.txData = srcBuff;
xfer.rxData = destBuff;
xfer.dataSize = BUFFER_SIZE;
xfer.flags = kFLEXIO_SPI_8bitMsb;
FLEXIO_SPI_MasterTransferCreateHandleDMA(&spiDev, &g_spiHandle,
    FLEXIO_SPI_MasterUserCallback, NULL, &g_spiTxDmaHandle, &g_spiRxDmaHandle);
FLEXIO_SPI_MasterTransferDMA(&spiDev, &g_spiHandle, &xfer);

// Send finished.
while (!txFinished)
{
}

// ...
}

```

## Modules

- [FlexIO DMA SPI Driver](#)
- [FlexIO eDMA SPI Driver](#)

## Data Structures

- struct [FLEXIO\\_SPI\\_Type](#)  
*Define FlexIO SPI access structure typedef.* [More...](#)
- struct [flexio\\_spi\\_master\\_config\\_t](#)  
*Define FlexIO SPI master configuration structure.* [More...](#)
- struct [flexio\\_spi\\_slave\\_config\\_t](#)  
*Define FlexIO SPI slave configuration structure.* [More...](#)

## FlexIO SPI Driver

- struct `flexio_spi_transfer_t`  
*Define FlexIO SPI transfer structure. [More...](#)*
- struct `flexio_spi_master_handle_t`  
*Define FlexIO SPI handle structure. [More...](#)*

## Macros

- #define `FLEXIO_SPI_DUMMYDATA` (0xFFFFU)  
*FlexIO SPI dummy transfer data, the data is sent while txData is NULL.*

## Typedefs

- typedef flexio\_spi\_master\_handle\_t `flexio_spi_slave_handle_t`  
*Slave handle is the same with master handle.*
- typedef void(\* `flexio_spi_master_transfer_callback_t` )(FLEXIO\_SPI\_Type \*base, flexio\_spi\_master\_handle\_t \*handle, status\_t status, void \*userData)  
*FlexIO SPI master callback for finished transmit.*
- typedef void(\* `flexio_spi_slave_transfer_callback_t` )(FLEXIO\_SPI\_Type \*base, `flexio_spi_slave_handle_t` \*handle, status\_t status, void \*userData)  
*FlexIO SPI slave callback for finished transmit.*

## Enumerations

- enum `_flexio_spi_status` {  
  kStatus\_FLEXIO\_SPI\_Busy = MAKE\_STATUS(kStatusGroup\_FLEXIO\_SPI, 1),  
  kStatus\_FLEXIO\_SPI\_Idle = MAKE\_STATUS(kStatusGroup\_FLEXIO\_SPI, 2),  
  kStatus\_FLEXIO\_SPI\_Error = MAKE\_STATUS(kStatusGroup\_FLEXIO\_SPI, 3) }  
*Error codes for the FlexIO SPI driver.*
- enum `flexio_spi_clock_phase_t` {  
  kFLEXIO\_SPI\_ClockPhaseFirstEdge = 0x0U,  
  kFLEXIO\_SPI\_ClockPhaseSecondEdge = 0x1U }  
*FlexIO SPI clock phase configuration.*
- enum `flexio_spi_shift_direction_t` {  
  kFLEXIO\_SPI\_MsbFirst = 0,  
  kFLEXIO\_SPI\_LsbFirst = 1 }  
*FlexIO SPI data shifter direction options.*
- enum `flexio_spi_data_bitcount_mode_t` {  
  kFLEXIO\_SPI\_8BitMode = 0x08U,  
  kFLEXIO\_SPI\_16BitMode = 0x10U }  
*FlexIO SPI data length mode options.*
- enum `_flexio_spi_interrupt_enable` {  
  kFLEXIO\_SPI\_TxEmptyInterruptEnable = 0x1U,  
  kFLEXIO\_SPI\_RxFullInterruptEnable = 0x2U }  
*Define FlexIO SPI interrupt mask.*

- enum \_flexio\_spi\_status\_flags {
 kFLEXIO\_SPI\_TxBufferEmptyFlag = 0x1U,
 kFLEXIO\_SPI\_RxBufferFullFlag = 0x2U }
   
*Define FlexIO SPI status mask.*
- enum \_flexio\_spi\_dma\_enable {
 kFLEXIO\_SPI\_TxDmaEnable = 0x1U,
 kFLEXIO\_SPI\_RxDmaEnable = 0x2U,
 kFLEXIO\_SPI\_DmaAllEnable = 0x3U }
   
*Define FlexIO SPI DMA mask.*
- enum \_flexio\_spi\_transfer\_flags {
 kFLEXIO\_SPI\_8bitMsb = 0x1U,
 kFLEXIO\_SPI\_8bitLsb = 0x2U,
 kFLEXIO\_SPI\_16bitMsb = 0x9U,
 kFLEXIO\_SPI\_16bitLsb = 0xaU }
   
*Define FlexIO SPI transfer flags.*

## Driver version

- #define FSL\_FLEXIO\_SPI\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 0))
   
*FlexIO SPI driver version 2.1.0.*

## FlexIO SPI Configuration

- void FLEXIO\_SPI\_MasterInit (FLEXIO\_SPI\_Type \*base, flexio\_spi\_master\_config\_t \*masterConfig, uint32\_t srcClock\_Hz)
   
*Ungates the FlexIO clock, resets the FlexIO module and configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration.*
- void FLEXIO\_SPI\_MasterDeinit (FLEXIO\_SPI\_Type \*base)
   
*Gates the FlexIO clock.*
- void FLEXIO\_SPI\_MasterGetDefaultConfig (flexio\_spi\_master\_config\_t \*masterConfig)
   
*Gets the default configuration to configure the FlexIO SPI master.*
- void FLEXIO\_SPI\_SlaveInit (FLEXIO\_SPI\_Type \*base, flexio\_spi\_slave\_config\_t \*slaveConfig)
   
*Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration.*
- void FLEXIO\_SPI\_SlaveDeinit (FLEXIO\_SPI\_Type \*base)
   
*Gates the FlexIO clock.*
- void FLEXIO\_SPI\_SlaveGetDefaultConfig (flexio\_spi\_slave\_config\_t \*slaveConfig)
   
*Gets the default configuration to configure the FlexIO SPI slave.*

## Status

- uint32\_t FLEXIO\_SPI\_GetStatusFlags (FLEXIO\_SPI\_Type \*base)
   
*Gets FlexIO SPI status flags.*
- void FLEXIO\_SPI\_ClearStatusFlags (FLEXIO\_SPI\_Type \*base, uint32\_t mask)
   
*Clears FlexIO SPI status flags.*

## FlexIO SPI Driver

### Interrupts

- void **FLEXIO\_SPI\_EnableInterrupts** (**FLEXIO\_SPI\_Type** \*base, **uint32\_t** mask)  
*Enables the FlexIO SPI interrupt.*
- void **FLEXIO\_SPI\_DisableInterrupts** (**FLEXIO\_SPI\_Type** \*base, **uint32\_t** mask)  
*Disables the FlexIO SPI interrupt.*

### DMA Control

- void **FLEXIO\_SPI\_EnableDMA** (**FLEXIO\_SPI\_Type** \*base, **uint32\_t** mask, **bool** enable)  
*Enables/disables the FlexIO SPI transmit DMA.*
- static **uint32\_t FLEXIO\_SPI\_GetTxDataRegisterAddress** (**FLEXIO\_SPI\_Type** \*base, **flexio\_spi\_shift\_direction\_t** direction)  
*Gets the FlexIO SPI transmit data register address for MSB first transfer.*
- static **uint32\_t FLEXIO\_SPI\_GetRxDataRegisterAddress** (**FLEXIO\_SPI\_Type** \*base, **flexio\_spi\_shift\_direction\_t** direction)  
*Gets the FlexIO SPI receive data register address for the MSB first transfer.*

### Bus Operations

- static void **FLEXIO\_SPI\_Enable** (**FLEXIO\_SPI\_Type** \*base, **bool** enable)  
*Enables/disables the FlexIO SPI module operation.*
- void **FLEXIO\_SPI\_MasterSetBaudRate** (**FLEXIO\_SPI\_Type** \*base, **uint32\_t** baudRate\_Bps, **uint32\_t** srcClockHz)  
*Sets baud rate for the FlexIO SPI transfer, which is only used for the master.*
- static void **FLEXIO\_SPI\_WriteData** (**FLEXIO\_SPI\_Type** \*base, **flexio\_spi\_shift\_direction\_t** direction, **uint16\_t** data)  
*Writes one byte of data, which is sent using the MSB method.*
- static **uint16\_t FLEXIO\_SPI\_ReadData** (**FLEXIO\_SPI\_Type** \*base, **flexio\_spi\_shift\_direction\_t** direction)  
*Reads 8 bit/16 bit data.*
- void **FLEXIO\_SPI\_WriteBlocking** (**FLEXIO\_SPI\_Type** \*base, **flexio\_spi\_shift\_direction\_t** direction, const **uint8\_t** \*buffer, **size\_t** size)  
*Sends a buffer of data bytes.*
- void **FLEXIO\_SPI\_ReadBlocking** (**FLEXIO\_SPI\_Type** \*base, **flexio\_spi\_shift\_direction\_t** direction, **uint8\_t** \*buffer, **size\_t** size)  
*Receives a buffer of bytes.*
- void **FLEXIO\_SPI\_MasterTransferBlocking** (**FLEXIO\_SPI\_Type** \*base, **flexio\_spi\_transfer\_t** \*xfer)  
*Receives a buffer of bytes.*

### Transactional

- **status\_t FLEXIO\_SPI\_MasterTransferCreateHandle** (**FLEXIO\_SPI\_Type** \*base, **flexio\_spi\_master\_handle\_t** \*handle, **flexio\_spi\_master\_transfer\_callback\_t** callback, **void** \*userData)  
*Initializes the FlexIO SPI Master handle, which is used in transactional functions.*

- status\_t **FLEXIO\_SPI\_MasterTransferNonBlocking** (FLEXIO\_SPI\_Type \*base, flexio\_spi\_master\_handle\_t \*handle, flexio\_spi\_transfer\_t \*xfer)  
*Master transfer data using IRQ.*
- void **FLEXIO\_SPI\_MasterTransferAbort** (FLEXIO\_SPI\_Type \*base, flexio\_spi\_master\_handle\_t \*handle)  
*Aborts the master data transfer, which used IRQ.*
- status\_t **FLEXIO\_SPI\_MasterTransferGetCount** (FLEXIO\_SPI\_Type \*base, flexio\_spi\_master\_handle\_t \*handle, size\_t \*count)  
*Gets the data transfer status which used IRQ.*
- void **FLEXIO\_SPI\_MasterTransferHandleIRQ** (void \*spiType, void \*spiHandle)  
*FlexIO SPI master IRQ handler function.*
- status\_t **FLEXIO\_SPI\_SlaveTransferCreateHandle** (FLEXIO\_SPI\_Type \*base, flexio\_spi\_slave\_handle\_t \*handle, flexio\_spi\_slave\_transfer\_callback\_t callback, void \*userData)  
*Initializes the FlexIO SPI Slave handle, which is used in transactional functions.*
- status\_t **FLEXIO\_SPI\_SlaveTransferNonBlocking** (FLEXIO\_SPI\_Type \*base, flexio\_spi\_slave\_handle\_t \*handle, flexio\_spi\_transfer\_t \*xfer)  
*Slave transfer data using IRQ.*
- static void **FLEXIO\_SPI\_SlaveTransferAbort** (FLEXIO\_SPI\_Type \*base, flexio\_spi\_slave\_handle\_t \*handle)  
*Aborts the slave data transfer which used IRQ, share same API with master.*
- static status\_t **FLEXIO\_SPI\_SlaveTransferGetCount** (FLEXIO\_SPI\_Type \*base, flexio\_spi\_slave\_handle\_t \*handle, size\_t \*count)  
*Gets the data transfer status which used IRQ, share same API with master.*
- void **FLEXIO\_SPI\_SlaveTransferHandleIRQ** (void \*spiType, void \*spiHandle)  
*FlexIO SPI slave IRQ handler function.*

## 15.6.3 Data Structure Documentation

### 15.6.3.1 struct FLEXIO\_SPI\_Type

#### Data Fields

- FLEXIO\_Type \* **flexioBase**  
*FlexIO base pointer.*
- uint8\_t **SDOPinIndex**  
*Pin select for data output.*
- uint8\_t **SDIPinIndex**  
*Pin select for data input.*
- uint8\_t **SCKPinIndex**  
*Pin select for clock.*
- uint8\_t **CSnPinIndex**  
*Pin select for enable.*
- uint8\_t **shifterIndex** [2]  
*Shifter index used in FlexIO SPI.*
- uint8\_t **timerIndex** [2]  
*Timer index used in FlexIO SPI.*

## FlexIO SPI Driver

### 15.6.3.1.0.1.1 Field Documentation

15.6.3.1.0.1.1 `FLEXIO_Type* FLEXIO_SPI_Type::flexioBase`

15.6.3.1.0.1.2 `uint8_t FLEXIO_SPI_Type::SDOPinIndex`

15.6.3.1.0.1.3 `uint8_t FLEXIO_SPI_Type::SDIPinIndex`

15.6.3.1.0.1.4 `uint8_t FLEXIO_SPI_Type::SCKPinIndex`

15.6.3.1.0.1.5 `uint8_t FLEXIO_SPI_Type::CSnPinIndex`

15.6.3.1.0.1.6 `uint8_t FLEXIO_SPI_Type::shifterIndex[2]`

15.6.3.1.0.1.7 `uint8_t FLEXIO_SPI_Type::timerIndex[2]`

### 15.6.3.2 struct flexio\_spi\_master\_config\_t

#### Data Fields

- bool `enableMaster`  
*Enable/disable FlexIO SPI master after configuration.*
- bool `enableInDoze`  
*Enable/disable FlexIO operation in doze mode.*
- bool `enableInDebug`  
*Enable/disable FlexIO operation in debug mode.*
- bool `enableFastAccess`  
*Enable/disable fast access to FlexIO registers,  
fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- `uint32_t baudRate_Bps`  
*Baud rate in Bps.*
- `flexio_spi_clock_phase_t phase`  
*Clock phase.*
- `flexio_spi_data_bitcount_mode_t dataMode`  
*8bit or 16bit mode.*

### 15.6.3.2.0.2 Field Documentation

15.6.3.2.0.2.1 `bool flexio_spi_master_config_t::enableMaster`

15.6.3.2.0.2.2 `bool flexio_spi_master_config_t::enableInDoze`

15.6.3.2.0.2.3 `bool flexio_spi_master_config_t::enableInDebug`

15.6.3.2.0.2.4 `bool flexio_spi_master_config_t::enableFastAccess`

15.6.3.2.0.2.5 `uint32_t flexio_spi_master_config_t::baudRate_Bps`

15.6.3.2.0.2.6 `flexio_spi_clock_phase_t flexio_spi_master_config_t::phase`

15.6.3.2.0.2.7 `flexio_spi_data_bitcount_mode_t flexio_spi_master_config_t::dataMode`

### 15.6.3.3 `struct flexio_spi_slave_config_t`

#### Data Fields

- `bool enableSlave`  
*Enable/disable FlexIO SPI slave after configuration.*
- `bool enableInDoze`  
*Enable/disable FlexIO operation in doze mode.*
- `bool enableInDebug`  
*Enable/disable FlexIO operation in debug mode.*
- `bool enableFastAccess`  
*Enable/disable fast access to FlexIO registers,  
fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- `flexio_spi_clock_phase_t phase`  
*Clock phase.*
- `flexio_spi_data_bitcount_mode_t dataMode`  
*8bit or 16bit mode.*

## FlexIO SPI Driver

### 15.6.3.3.0.3 Field Documentation

15.6.3.3.0.3.1 `bool flexio_spi_slave_config_t::enableSlave`

15.6.3.3.0.3.2 `bool flexio_spi_slave_config_t::enableInDoze`

15.6.3.3.0.3.3 `bool flexio_spi_slave_config_t::enableInDebug`

15.6.3.3.0.3.4 `bool flexio_spi_slave_config_t::enableFastAccess`

15.6.3.3.0.3.5 `flexio_spi_clock_phase_t flexio_spi_slave_config_t::phase`

15.6.3.3.0.3.6 `flexio_spi_data_bitcount_mode_t flexio_spi_slave_config_t::dataMode`

### 15.6.3.4 `struct flexio_spi_transfer_t`

#### Data Fields

- `uint8_t * txData`  
*Send buffer.*
- `uint8_t * rxData`  
*Receive buffer.*
- `size_t dataSize`  
*Transfer bytes.*
- `uint8_t flags`  
*FlexIO SPI control flag, MSB first or LSB first.*

### 15.6.3.4.0.4 Field Documentation

15.6.3.4.0.4.1 `uint8_t* flexio_spi_transfer_t::txData`

15.6.3.4.0.4.2 `uint8_t* flexio_spi_transfer_t::rxData`

15.6.3.4.0.4.3 `size_t flexio_spi_transfer_t::dataSize`

15.6.3.4.0.4.4 `uint8_t flexio_spi_transfer_t::flags`

### 15.6.3.5 `struct _flexio_spi_master_handle`

typedef for `flexio_spi_master_handle_t` in advance.

#### Data Fields

- `uint8_t * txData`  
*Transfer buffer.*
- `uint8_t * rxData`  
*Receive buffer.*
- `size_t transferSize`  
*Total bytes to be transferred.*
- `volatile size_t txRemainingBytes`

- volatile size\_t **rxRemainingBytes**  
*Send data remaining in bytes.*
- volatile uint32\_t **state**  
*Receive data remaining in bytes.*
- uint8\_t **bytePerFrame**  
*FlexIO SPI internal state.*
- flexio\_spi\_shift\_direction\_t **direction**  
*SPI mode, 2bytes or 1byte in a frame.*
- flexio\_spi\_master\_transfer\_callback\_t **callback**  
*Shift direction.*
- void \* **userData**  
*FlexIO SPI callback.*
- Callback parameter.

## FlexIO SPI Driver

### 15.6.3.5.0.5 Field Documentation

15.6.3.5.0.5.1 `uint8_t* flexio_spi_master_handle_t::txData`

15.6.3.5.0.5.2 `uint8_t* flexio_spi_master_handle_t::rxData`

15.6.3.5.0.5.3 `size_t flexio_spi_master_handle_t::transferSize`

15.6.3.5.0.5.4 `volatile size_t flexio_spi_master_handle_t::txRemainingBytes`

15.6.3.5.0.5.5 `volatile size_t flexio_spi_master_handle_t::rxRemainingBytes`

15.6.3.5.0.5.6 `volatile uint32_t flexio_spi_master_handle_t::state`

15.6.3.5.0.5.7 `flexio_spi_shift_direction_t flexio_spi_master_handle_t::direction`

15.6.3.5.0.5.8 `flexio_spi_master_transfer_callback_t flexio_spi_master_handle_t::callback`

15.6.3.5.0.5.9 `void* flexio_spi_master_handle_t::userData`

### 15.6.4 Macro Definition Documentation

15.6.4.1 `#define FSL_FLEXIO_SPI_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

15.6.4.2 `#define FLEXIO_SPI_DUMMYDATA (0xFFFFU)`

### 15.6.5 Typedef Documentation

15.6.5.1 `typedef flexio_spi_master_handle_t flexio_spi_slave_handle_t`

### 15.6.6 Enumeration Type Documentation

#### 15.6.6.1 `enum _flexio_spi_status`

Enumerator

*kStatus\_FLEXIO\_SPI\_Busy* FlexIO SPI is busy.

*kStatus\_FLEXIO\_SPI\_Idle* SPI is idle.

*kStatus\_FLEXIO\_SPI\_Error* FlexIO SPI error.

#### 15.6.6.2 `enum flexio_spi_clock_phase_t`

Enumerator

*kFLEXIO\_SPI\_ClockPhaseFirstEdge* First edge on SPSCK occurs at the middle of the first cycle of a data transfer.

***kFLEXIO\_SPI\_ClockPhaseSecondEdge*** First edge on SPSCK occurs at the start of the first cycle of a data transfer.

#### 15.6.6.3 enum flexio\_spi\_shift\_direction\_t

Enumerator

***kFLEXIO\_SPI\_MsbFirst*** Data transfers start with most significant bit.

***kFLEXIO\_SPI\_LsbFirst*** Data transfers start with least significant bit.

#### 15.6.6.4 enum flexio\_spi\_data\_bitcount\_mode\_t

Enumerator

***kFLEXIO\_SPI\_8BitMode*** 8-bit data transmission mode.

***kFLEXIO\_SPI\_16BitMode*** 16-bit data transmission mode.

#### 15.6.6.5 enum \_flexio\_spi\_interrupt\_enable

Enumerator

***kFLEXIO\_SPI\_TxEmptyInterruptEnable*** Transmit buffer empty interrupt enable.

***kFLEXIO\_SPI\_RxFullInterruptEnable*** Receive buffer full interrupt enable.

#### 15.6.6.6 enum \_flexio\_spi\_status\_flags

Enumerator

***kFLEXIO\_SPI\_TxBufferEmptyFlag*** Transmit buffer empty flag.

***kFLEXIO\_SPI\_RxBufferFullFlag*** Receive buffer full flag.

#### 15.6.6.7 enum \_flexio\_spi\_dma\_enable

Enumerator

***kFLEXIO\_SPI\_TxDmaEnable*** Tx DMA request source.

***kFLEXIO\_SPI\_RxDmaEnable*** Rx DMA request source.

***kFLEXIO\_SPI\_DmaAllEnable*** All DMA request source.

## FlexIO SPI Driver

### 15.6.6.8 enum \_flexio\_spi\_transfer\_flags

Enumerator

**kFLEXIO\_SPI\_8bitMsb** FlexIO SPI 8-bit MSB first.  
**kFLEXIO\_SPI\_8bitLsb** FlexIO SPI 8-bit LSB first.  
**kFLEXIO\_SPI\_16bitMsb** FlexIO SPI 16-bit MSB first.  
**kFLEXIO\_SPI\_16bitLsb** FlexIO SPI 16-bit LSB first.

### 15.6.7 Function Documentation

#### 15.6.7.1 void FLEXIO\_SPI\_MasterInit ( **FLEXIO\_SPI\_Type** \* *base*,                           **flexio\_spi\_master\_config\_t** \* *masterConfig*, **uint32\_t** *srcClock\_Hz* )

The configuration structure can be filled by the user, or be set with default values by the [FLEXIO\\_SPI\\_MasterGetDefaultConfig\(\)](#).

Note

FlexIO SPI master only support CPOL = 0, which means clock inactive low.

Example

```
FLEXIO_SPI_Type spiDev = {  
.flexioBase = FLEXIO,  
.SDOPinIndex = 0,  
.SDIPinIndex = 1,  
.SCKPinIndex = 2,  
.CSnPinIndex = 3,  
.shifterIndex = {0,1},  
.timerIndex = {0,1}  
};  
flexio_spi_master_config_t config = {  
.enableMaster = true,  
.enableInDoze = false,  
.enableInDebug = true,  
.enableFastAccess = false,  
.baudRate_Bps = 500000,  
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,  
.direction = kFLEXIO_SPI_MsbFirst,  
.dataMode = kFLEXIO_SPI_8BitMode  
};  
FLEXIO_SPI_MasterInit(&spiDev, &config, srcClock_Hz);
```

Parameters

|                     |                                                                      |
|---------------------|----------------------------------------------------------------------|
| <i>base</i>         | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.            |
| <i>masterConfig</i> | Pointer to the <a href="#">flexio_spi_master_config_t</a> structure. |
| <i>srcClock_Hz</i>  | FlexIO source clock in Hz.                                           |

### 15.6.7.2 void FLEXIO\_SPI\_MasterDeinit ( [FLEXIO\\_SPI\\_Type](#) \* *base* )

Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_SPI_Type</a> . |
|-------------|--------------------------------------------------|

### 15.6.7.3 void FLEXIO\_SPI\_MasterGetDefaultConfig ( [flexio\\_spi\\_master\\_config\\_t](#) \* *masterConfig* )

The configuration can be used directly by calling the [FLEXIO\\_SPI\\_MasterConfigure\(\)](#). Example:

```
flexio_spi_master_config_t masterConfig;
FLEXIO\_SPI\_MasterGetDefaultConfig(&masterConfig);
```

Parameters

|                     |                                                                      |
|---------------------|----------------------------------------------------------------------|
| <i>masterConfig</i> | Pointer to the <a href="#">flexio_spi_master_config_t</a> structure. |
|---------------------|----------------------------------------------------------------------|

### 15.6.7.4 void FLEXIO\_SPI\_SlaveInit ( [FLEXIO\\_SPI\\_Type](#) \* *base*, [flexio\\_spi\\_slave\\_config\\_t](#) \* *slaveConfig* )

The configuration structure can be filled by the user, or be set with default values by the [FLEXIO\\_SPI\\_SlaveGetDefaultConfig\(\)](#).

Note

Only one timer is needed in the FlexIO SPI slave. As a result, the second timer index is ignored. FlexIO SPI slave only support CPOL = 0, which means clock inactive low. Example

```
FLEXIO_SPI_Type spiDev = {
    .flexioBase = FLEXIO,
    .SDOPinIndex = 0,
    .SDIPinIndex = 1,
    .SCKPinIndex = 2,
    .CSnPinIndex = 3,
    .shifterIndex = {0,1},
    .timerIndex = {0}
};
flexio_spi_slave_config_t config = {
    .enableSlave = true,
    .enableInDoze = false,
```

## FlexIO SPI Driver

```
.enableInDebug = true,  
.enableFastAccess = false,  
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,  
.direction = kFLEXIO_SPI_MsbFirst,  
.dataMode = kFLEXIO_SPI_8BitMode  
};  
FLEXIO_SPI_SlaveInit(&spiDev, &config);
```

Parameters

|                    |                                                            |
|--------------------|------------------------------------------------------------|
| <i>base</i>        | Pointer to the <b>FLEXIO_SPI_Type</b> structure.           |
| <i>slaveConfig</i> | Pointer to the <b>flexio_spi_slave_config_t</b> structure. |

### 15.6.7.5 void **FLEXIO\_SPI\_SlaveDeinit** ( **FLEXIO\_SPI\_Type** \* *base* )

Parameters

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

### 15.6.7.6 void **FLEXIO\_SPI\_SlaveGetDefaultConfig** ( **flexio\_spi\_slave\_config\_t** \* *slaveConfig* )

The configuration can be used directly for calling the **FLEXIO\_SPI\_SlaveConfigure()**. Example:

```
flexio_spi_slave_config_t slaveConfig;  
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

|                    |                                                            |
|--------------------|------------------------------------------------------------|
| <i>slaveConfig</i> | Pointer to the <b>flexio_spi_slave_config_t</b> structure. |
|--------------------|------------------------------------------------------------|

### 15.6.7.7 uint32\_t **FLEXIO\_SPI\_GetStatusFlags** ( **FLEXIO\_SPI\_Type** \* *base* )

Parameters

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

Returns

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

- kFLEXIO\_SPI\_TxEmptyFlag
- kFLEXIO\_SPI\_RxEmptyFlag

```
15.6.7.8 void FLEXIO_SPI_ClearStatusFlags ( FLEXIO_SPI_Type * base, uint32_t mask
    )
```

## FlexIO SPI Driver

Parameters

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

**15.6.7.9 void FLEXIO\_SPI\_EnableInterrupts ( FLEXIO\_SPI\_Type \* *base*, uint32\_t *mask* )**

This function enables the FlexIO SPI interrupt.

Parameters

|             |                                                                                                                                                                                                                  |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                                                                                                                                        |
| <i>mask</i> | interrupt source. The parameter can be any combination of the following values: <ul style="list-style-type: none"><li>• kFLEXIO_SPI_RxFullInterruptEnable</li><li>• kFLEXIO_SPI_TxEmptyInterruptEnable</li></ul> |

**15.6.7.10 void FLEXIO\_SPI\_DisableInterrupts ( FLEXIO\_SPI\_Type \* *base*, uint32\_t *mask* )**

This function disables the FlexIO SPI interrupt.

Parameters

|             |                                                                                                                                                                                                                 |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                                                                                                                                       |
| <i>mask</i> | interrupt source The parameter can be any combination of the following values: <ul style="list-style-type: none"><li>• kFLEXIO_SPI_RxFullInterruptEnable</li><li>• kFLEXIO_SPI_TxEmptyInterruptEnable</li></ul> |

**15.6.7.11 void FLEXIO\_SPI\_EnableDMA ( FLEXIO\_SPI\_Type \* *base*, uint32\_t *mask*, bool *enable* )**

This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the kFLEXIO\_SPI\_TxEmptyFlag does/doesn't trigger the DMA request.

Parameters

|               |                                                           |
|---------------|-----------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>mask</i>   | SPI DMA source.                                           |
| <i>enable</i> | True means enable DMA, false means disable DMA.           |

#### **15.6.7.12 static uint32\_t FLEXIO\_SPI\_GetTxDataRegisterAddress ( FLEXIO\_SPI\_Type \* *base*, flexio\_spi\_shift\_direction\_t *direction* ) [inline], [static]**

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>direction</i> | Shift direction of MSB first or LSB first.                |

Returns

FlexIO SPI transmit data register address.

#### **15.6.7.13 static uint32\_t FLEXIO\_SPI\_GetRxDataRegisterAddress ( FLEXIO\_SPI\_Type \* *base*, flexio\_spi\_shift\_direction\_t *direction* ) [inline], [static]**

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>direction</i> | Shift direction of MSB first or LSB first.                |

Returns

FlexIO SPI receive data register address.

#### **15.6.7.14 static void FLEXIO\_SPI\_Enable ( FLEXIO\_SPI\_Type \* *base*, bool *enable* ) [inline], [static]**

## FlexIO SPI Driver

Parameters

|               |                                                  |
|---------------|--------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> . |
| <i>enable</i> | True to enable, false to disable.                |

**15.6.7.15 void FLEXIO\_SPI\_MasterSetBaudRate ( [FLEXIO\\_SPI\\_Type](#) \* *base*, [uint32\\_t](#) *baudRate\_Bps*, [uint32\\_t](#) *srcClockHz* )**

Parameters

|                     |                                                           |
|---------------------|-----------------------------------------------------------|
| <i>base</i>         | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>baudRate_Bps</i> | Baud Rate needed in Hz.                                   |
| <i>srcClockHz</i>   | SPI source clock frequency in Hz.                         |

**15.6.7.16 static void FLEXIO\_SPI\_WriteData ( [FLEXIO\\_SPI\\_Type](#) \* *base*, [flexio\\_spi\\_shift\\_direction\\_t](#) *direction*, [uint16\\_t](#) *data* ) [inline], [static]**

Note

This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>direction</i> | Shift direction of MSB first or LSB first.                |
| <i>data</i>      | 8 bit/16 bit data.                                        |

**15.6.7.17 static [uint16\\_t](#) FLEXIO\_SPI\_ReadData ( [FLEXIO\\_SPI\\_Type](#) \* *base*, [flexio\\_spi\\_shift\\_direction\\_t](#) *direction* ) [inline], [static]**

Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>direction</i> | Shift direction of MSB first or LSB first.                |

Returns

8 bit/16 bit data received.

**15.6.7.18 void FLEXIO\_SPI\_WriteBlocking ( FLEXIO\_SPI\_Type \* *base*, flexio\_spi\_shift\_direction\_t *direction*, const uint8\_t \* *buffer*, size\_t *size* )**

Note

This function blocks using the polling method until all bytes have been sent.

Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>direction</i> | Shift direction of MSB first or LSB first.                |
| <i>buffer</i>    | The data bytes to send.                                   |
| <i>size</i>      | The number of data bytes to send.                         |

**15.6.7.19 void FLEXIO\_SPI\_ReadBlocking ( FLEXIO\_SPI\_Type \* *base*, flexio\_spi\_shift\_direction\_t *direction*, uint8\_t \* *buffer*, size\_t *size* )**

Note

This function blocks using the polling method until all bytes have been received.

Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>direction</i> | Shift direction of MSB first or LSB first.                |
| <i>buffer</i>    | The buffer to store the received bytes.                   |

## FlexIO SPI Driver

|                  |                                            |
|------------------|--------------------------------------------|
| <i>size</i>      | The number of data bytes to be received.   |
| <i>direction</i> | Shift direction of MSB first or LSB first. |

**15.6.7.20 void FLEXIO\_SPI\_MasterTransferBlocking ( FLEXIO\_SPI\_Type \* *base*, flexio\_spi\_transfer\_t \* *xfer* )**

Note

This function blocks via polling until all bytes have been received.

Parameters

|             |                                                                            |
|-------------|----------------------------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_SPI_Type</a> structure                       |
| <i>xfer</i> | FlexIO SPI transfer structure, see <a href="#">flexio_spi_transfer_t</a> . |

**15.6.7.21 status\_t FLEXIO\_SPI\_MasterTransferCreateHandle ( FLEXIO\_SPI\_Type \* *base*, flexio\_spi\_master\_handle\_t \* *handle*, flexio\_spi\_master\_transfer\_callback\_t *callback*, void \* *userData* )**

Parameters

|                 |                                                                                                  |
|-----------------|--------------------------------------------------------------------------------------------------|
| <i>base</i>     | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                        |
| <i>handle</i>   | Pointer to the <a href="#">flexio_spi_master_handle_t</a> structure to store the transfer state. |
| <i>callback</i> | The callback function.                                                                           |
| <i>userData</i> | The parameter of the callback function.                                                          |

Return values

|                           |                                                |
|---------------------------|------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                |
| <i>kStatus_OutOfRange</i> | The FlexIO type/handle/ISR table out of range. |

**15.6.7.22 status\_t FLEXIO\_SPI\_MasterTransferNonBlocking ( FLEXIO\_SPI\_Type \* *base*, flexio\_spi\_master\_handle\_t \* *handle*, flexio\_spi\_transfer\_t \* *xfer* )**

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                     |
| <i>handle</i> | Pointer to the <code>flexio_spi_master_handle_t</code> structure to store the transfer state. |
| <i>xfer</i>   | FlexIO SPI transfer structure. See <a href="#">flexio_spi_transfer_t</a> .                    |

Return values

|                                |                                               |
|--------------------------------|-----------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                    |
| <i>kStatus_FLEXIO_SPI_Busy</i> | SPI is not idle, is running another transfer. |

#### 15.6.7.23 void FLEXIO\_SPI\_MasterTransferAbort ( `FLEXIO_SPI_Type * base,` `flexio_spi_master_handle_t * handle` )

Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                     |
| <i>handle</i> | Pointer to the <code>flexio_spi_master_handle_t</code> structure to store the transfer state. |

#### 15.6.7.24 status\_t FLEXIO\_SPI\_MasterTransferGetCount ( `FLEXIO_SPI_Type * base,` `flexio_spi_master_handle_t * handle, size_t * count` )

Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                     |
| <i>handle</i> | Pointer to the <code>flexio_spi_master_handle_t</code> structure to store the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.                           |

Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_InvalidArgument</i> | count is Invalid.              |
| <i>kStatus_Success</i>         | Successfully return the count. |

#### 15.6.7.25 void FLEXIO\_SPI\_MasterTransferHandleIRQ ( `void * spiType, void * spiHandle` `)`

## FlexIO SPI Driver

Parameters

|                  |                                                                                                  |
|------------------|--------------------------------------------------------------------------------------------------|
| <i>spiType</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                        |
| <i>spiHandle</i> | Pointer to the <a href="#">flexio_spi_master_handle_t</a> structure to store the transfer state. |

**15.6.7.26 status\_t FLEXIO\_SPI\_SlaveTransferCreateHandle ( [FLEXIO\\_SPI\\_Type](#) \* *base*, [flexio\\_spi\\_slave\\_handle\\_t](#) \* *handle*, [flexio\\_spi\\_slave\\_transfer\\_callback\\_t](#) *callback*, [void](#) \* *userData* )**

Parameters

|                 |                                                                                                 |
|-----------------|-------------------------------------------------------------------------------------------------|
| <i>base</i>     | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                       |
| <i>handle</i>   | Pointer to the <a href="#">flexio_spi_slave_handle_t</a> structure to store the transfer state. |
| <i>callback</i> | The callback function.                                                                          |
| <i>userData</i> | The parameter of the callback function.                                                         |

Return values

|                           |                                                |
|---------------------------|------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                |
| <i>kStatus_OutOfRange</i> | The FlexIO type/handle/ISR table out of range. |

**15.6.7.27 status\_t FLEXIO\_SPI\_SlaveTransferNonBlocking ( [FLEXIO\\_SPI\\_Type](#) \* *base*, [flexio\\_spi\\_slave\\_handle\\_t](#) \* *handle*, [flexio\\_spi\\_transfer\\_t](#) \* *xfer* )**

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

|               |                                                                                                 |
|---------------|-------------------------------------------------------------------------------------------------|
| <i>handle</i> | Pointer to the <a href="#">flexio_spi_slave_handle_t</a> structure to store the transfer state. |
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                       |
| <i>xfer</i>   | FlexIO SPI transfer structure. See <a href="#">flexio_spi_transfer_t</a> .                      |

Return values

|                                |                                                  |
|--------------------------------|--------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                   |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                       |
| <i>kStatus_FLEXIO_SPI_Busy</i> | SPI is not idle; it is running another transfer. |

15.6.7.28 **static void FLEXIO\_SPI\_SlaveTransferAbort( FLEXIO\_SPI\_Type \* *base*, flexio\_spi\_slave\_handle\_t \* *handle* ) [inline], [static]**

## FlexIO SPI Driver

Parameters

|               |                                                                                 |
|---------------|---------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                       |
| <i>handle</i> | Pointer to the flexio_spi_slave_handle_t structure to store the transfer state. |

**15.6.7.29 static status\_t FLEXIO\_SPI\_SlaveTransferGetCount ( FLEXIO\_SPI\_Type \* *base*, flexio\_spi\_slave\_handle\_t \* *handle*, size\_t \* *count* ) [inline], [static]**

Parameters

|               |                                                                                 |
|---------------|---------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                       |
| <i>handle</i> | Pointer to the flexio_spi_slave_handle_t structure to store the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.             |

Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_InvalidArgument</i> | count is Invalid.              |
| <i>kStatus_Success</i>         | Successfully return the count. |

**15.6.7.30 void FLEXIO\_SPI\_SlaveTransferHandleIRQ ( void \* *spiType*, void \* *spiHandle* )**

Parameters

|                  |                                                                                 |
|------------------|---------------------------------------------------------------------------------|
| <i>spiType</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                       |
| <i>spiHandle</i> | Pointer to the flexio_spi_slave_handle_t structure to store the transfer state. |

## 15.6.8 FlexIO eDMA SPI Driver

### 15.6.8.1 Overview

#### Data Structures

- struct `flexio_spi_master_edma_handle_t`

*FlexIO SPI eDMA transfer handle, users should not touch the content of the handle. [More...](#)*

#### TypeDefs

- typedef `flexio_spi_master_edma_handle_t flexio_spi_slave_edma_handle_t`  
*Slave handle is the same with master handle.*
- typedef void(\* `flexio_spi_master_edma_transfer_callback_t`)(`FLEXIO_SPI_Type` \*base, `flexio_spi_master_edma_handle_t` \*handle, `status_t` status, `void` \*userData)  
*FlexIO SPI master callback for finished transmit.*
- typedef void(\* `flexio_spi_slave_edma_transfer_callback_t`)(`FLEXIO_SPI_Type` \*base, `flexio_spi_slave_edma_handle_t` \*handle, `status_t` status, `void` \*userData)  
*FlexIO SPI slave callback for finished transmit.*

#### eDMA Transactional

- `status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA` (`FLEXIO_SPI_Type` \*base, `flexio_spi_master_edma_handle_t` \*handle, `flexio_spi_master_edma_transfer_callback_t` callback, `void` \*userData, `edma_handle_t` \*txHandle, `edma_handle_t` \*rxHandle)  
*Initializes the FLEXIO SPI master eDMA handle.*
- `status_t FLEXIO_SPI_MasterTransferEDMA` (`FLEXIO_SPI_Type` \*base, `flexio_spi_master_edma_handle_t` \*handle, `flexio_spi_transfer_t` \*xfer)  
*Performs a non-blocking FlexIO SPI transfer using eDMA.*
- `void FLEXIO_SPI_MasterTransferAbortEDMA` (`FLEXIO_SPI_Type` \*base, `flexio_spi_master_edma_handle_t` \*handle)  
*Aborts a FlexIO SPI transfer using eDMA.*
- `status_t FLEXIO_SPI_MasterTransferGetCountEDMA` (`FLEXIO_SPI_Type` \*base, `flexio_spi_master_edma_handle_t` \*handle, `size_t` \*count)  
*Gets the remaining bytes for FlexIO SPI eDMA transfer.*
- static void `FLEXIO_SPI_SlaveTransferCreateHandleEDMA` (`FLEXIO_SPI_Type` \*base, `flexio_spi_slave_edma_handle_t` \*handle, `flexio_spi_slave_edma_transfer_callback_t` callback, `void` \*userData, `edma_handle_t` \*txHandle, `edma_handle_t` \*rxHandle)  
*Initializes the FlexIO SPI slave eDMA handle.*
- `status_t FLEXIO_SPI_SlaveTransferEDMA` (`FLEXIO_SPI_Type` \*base, `flexio_spi_slave_edma_handle_t` \*handle, `flexio_spi_transfer_t` \*xfer)  
*Performs a non-blocking FlexIO SPI transfer using eDMA.*
- static void `FLEXIO_SPI_SlaveTransferAbortEDMA` (`FLEXIO_SPI_Type` \*base, `flexio_spi_slave_edma_handle_t` \*handle)  
*Aborts a FlexIO SPI transfer using eDMA.*

## FlexIO SPI Driver

- static status\_t **FLEXIO\_SPI\_SlaveTransferGetCountEDMA** (**FLEXIO\_SPI\_Type** \*base, **flexio\_spi\_slave\_edma\_handle\_t** \*handle, **size\_t** \*count)  
*Gets the remaining bytes to be transferred for FlexIO SPI eDMA.*

### 15.6.8.2 Data Structure Documentation

#### 15.6.8.2.1 **struct \_flexio\_spi\_master\_edma\_handle**

typedef for **flexio\_spi\_master\_edma\_handle\_t** in advance.

##### Data Fields

- **size\_t transferSize**  
*Total bytes to be transferred.*
- **bool txInProgress**  
*Send transfer in progress.*
- **bool rxInProgress**  
*Receive transfer in progress.*
- **edma\_handle\_t \* txHandle**  
*DMA handler for SPI send.*
- **edma\_handle\_t \* rxHandle**  
*DMA handler for SPI receive.*
- **flexio\_spi\_master\_edma\_transfer\_callback\_t callback**  
*Callback for SPI DMA transfer.*
- **void \* userData**  
*User Data for SPI DMA callback.*

#### 15.6.8.2.1.1 Field Documentation

##### 15.6.8.2.1.1.1 **size\_t flexio\_spi\_master\_edma\_handle\_t::transferSize**

### 15.6.8.3 Typedef Documentation

#### 15.6.8.3.1 **typedef flexio\_spi\_master\_edma\_handle\_t flexio\_spi\_slave\_edma\_handle\_t**

### 15.6.8.4 Function Documentation

#### 15.6.8.4.1 **status\_t FLEXIO\_SPI\_MasterTransferCreateHandleEDMA ( FLEXIO\_SPI\_Type \* base, flexio\_spi\_master\_edma\_handle\_t \* handle, flexio\_spi\_master\_edma\_transfer\_callback\_t callback, void \* userData, edma\_handle\_t \* txHandle, edma\_handle\_t \* rxHandle )**

This function initializes the FLEXO SPI master eDMA handle which can be used for other FLEXO SPI master transactional APIs. For a specified FLEXO SPI instance, call this API once to get the initialized handle.

## Parameters

|                 |                                                                                                |
|-----------------|------------------------------------------------------------------------------------------------|
| <i>base</i>     | pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                          |
| <i>handle</i>   | pointer to <code>flexio_spi_master_edma_handle_t</code> structure to store the transfer state. |
| <i>callback</i> | SPI callback, NULL means no callback.                                                          |
| <i>userData</i> | callback function parameter.                                                                   |
| <i>txHandle</i> | User requested eDMA handle for FlexIO SPI RX eDMA transfer.                                    |
| <i>rxHandle</i> | User requested eDMA handle for FlexIO SPI TX eDMA transfer.                                    |

## Return values

|                           |                                                     |
|---------------------------|-----------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                     |
| <i>kStatus_OutOfRange</i> | The FlexIO SPI eDMA type/handle table out of range. |

**15.6.8.4.2 status\_t FLEXIO\_SPI\_MasterTransferEDMA ( `FLEXIO_SPI_Type * base,`  
`flexio_spi_master_edma_handle_t * handle, flexio_spi_transfer_t * xfer` )**

## Note

This interface returns immediately after transfer initiates. Call `FLEXIO_SPI_MasterGetTransferCountEDMA` to poll the transfer status to check whether FlexIO SPI transfer finished.

## Parameters

|               |                                                                                                |
|---------------|------------------------------------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                          |
| <i>handle</i> | pointer to <code>flexio_spi_master_edma_handle_t</code> structure to store the transfer state. |
| <i>xfer</i>   | Pointer to FlexIO SPI transfer structure.                                                      |

## Return values

|                                |                                                      |
|--------------------------------|------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                       |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                           |
| <i>kStatus_FLEXIO_SPI_Busy</i> | FlexIO SPI is not idle, is running another transfer. |

**15.6.8.4.3 void FLEXIO\_SPI\_MasterTransferAbortEDMA ( `FLEXIO_SPI_Type * base,`  
`flexio_spi_master_edma_handle_t * handle` )**

## FlexIO SPI Driver

Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>handle</i> | FlexIO SPI eDMA handle pointer.                       |

### **15.6.8.4.4 status\_t FLEXIO\_SPI\_MasterTransferGetCountEDMA ( [FLEXIO\\_SPI\\_Type](#) \* *base*, [flexio\\_spi\\_master\\_edma\\_handle\\_t](#) \* *handle*, [size\\_t](#) \* *count* )**

Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_SPI_Type</a> structure.               |
| <i>handle</i> | FlexIO SPI eDMA handle pointer.                                     |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

### **15.6.8.4.5 static void FLEXIO\_SPI\_SlaveTransferCreateHandleEDMA ( [FLEXIO\\_SPI\\_Type](#) \* *base*, [flexio\\_spi\\_slave\\_edma\\_handle\\_t](#) \* *handle*, [flexio\\_spi\\_slave\\_edma\\_transfer\\_callback\\_t](#) *callback*, [void](#) \* *userData*, [edma\\_handle\\_t](#) \* *txHandle*, [edma\\_handle\\_t](#) \* *rxHandle* ) [inline], [**static**]**

This function initializes the FlexIO SPI slave eDMA handle.

Parameters

|                 |                                                                                                  |
|-----------------|--------------------------------------------------------------------------------------------------|
| <i>base</i>     | pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                            |
| <i>handle</i>   | pointer to <a href="#">flexio_spi_slave_edma_handle_t</a> structure to store the transfer state. |
| <i>callback</i> | SPI callback, NULL means no callback.                                                            |
| <i>userData</i> | callback function parameter.                                                                     |
| <i>txHandle</i> | User requested eDMA handle for FlexIO SPI TX eDMA transfer.                                      |
| <i>rxHandle</i> | User requested eDMA handle for FlexIO SPI RX eDMA transfer.                                      |

### **15.6.8.4.6 status\_t FLEXIO\_SPI\_SlaveTransferEDMA ( [FLEXIO\\_SPI\\_Type](#) \* *base*, [flexio\\_spi\\_slave\\_edma\\_handle\\_t](#) \* *handle*, [flexio\\_spi\\_transfer\\_t](#) \* *xfer* )**

Note

This interface returns immediately after transfer initiates. Call [FLEXIO\\_SPI\\_SlaveGetTransferCountEDMA](#) to poll the transfer status to check whether FlexIO SPI transfer finished.

Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                         |
| <i>handle</i> | pointer to <code>flexio_spi_slave_edma_handle_t</code> structure to store the transfer state. |
| <i>xfer</i>   | Pointer to FlexIO SPI transfer structure.                                                     |

Return values

|                                |                                                      |
|--------------------------------|------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                       |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                           |
| <i>kStatus_FLEXIO_SPI_Busy</i> | FlexIO SPI is not idle, is running another transfer. |

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

Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                         |
| <i>handle</i> | pointer to <code>flexio_spi_slave_edma_handle_t</code> structure to store the transfer state. |

#### 15.6.8.4.8 `static status_t FLEXIO_SPI_SlaveTransferGetCountEDMA ( FLEXIO_SPI_Type * base, flexio_spi_slave_edma_handle_t * handle, size_t * count ) [inline], [static]`

Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_SPI_Type</a> structure.               |
| <i>handle</i> | FlexIO SPI eDMA handle pointer.                                     |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

### 15.6.9 FlexIO DMA SPI Driver

#### 15.6.9.1 Overview

#### Data Structures

- struct `flexio_spi_master_dma_handle_t`

*FlexIO SPI DMA transfer handle, users should not touch the content of the handle.* [More...](#)

#### TypeDefs

- typedef `flexio_spi_master_dma_handle_t flexio_spi_slave_dma_handle_t`  
*Slave handle is the same with master handle.*
- typedef void(\* `flexio_spi_master_dma_transfer_callback_t`)(`FLEXIO_SPI_Type` \*base, `flexio_spi_master_dma_handle_t` \*handle, `status_t` status, `void` \*userData)  
*FlexIO SPI master callback for finished transmit.*
- typedef void(\* `flexio_spi_slave_dma_transfer_callback_t`)(`FLEXIO_SPI_Type` \*base, `flexio_spi_slave_dma_handle_t` \*handle, `status_t` status, `void` \*userData)  
*FlexIO SPI slave callback for finished transmit.*

#### DMA Transactional

- `status_t FLEXIO_SPI_MasterTransferCreateHandleDMA` (`FLEXIO_SPI_Type` \*base, `flexio_spi_master_dma_handle_t` \*handle, `flexio_spi_master_dma_transfer_callback_t` callback, `void` \*userData, `dma_handle_t` \*txHandle, `dma_handle_t` \*rxHandle)  
*Initializes the FLEXIO SPI master DMA handle.*
- `status_t FLEXIO_SPI_MasterTransferDMA` (`FLEXIO_SPI_Type` \*base, `flexio_spi_master_dma_handle_t` \*handle, `flexio_spi_transfer_t` \*xfer)  
*Performs a non-blocking FlexIO SPI transfer using DMA.*
- `void FLEXIO_SPI_MasterTransferAbortDMA` (`FLEXIO_SPI_Type` \*base, `flexio_spi_master_dma_handle_t` \*handle)  
*Aborts a FlexIO SPI transfer using DMA.*
- `status_t FLEXIO_SPI_MasterTransferGetCountDMA` (`FLEXIO_SPI_Type` \*base, `flexio_spi_master_dma_handle_t` \*handle, `size_t` \*count)  
*Gets the remaining bytes for FlexIO SPI DMA transfer.*
- `static void FLEXIO_SPI_SlaveTransferCreateHandleDMA` (`FLEXIO_SPI_Type` \*base, `flexio_spi_slave_dma_handle_t` \*handle, `flexio_spi_slave_dma_transfer_callback_t` callback, `void` \*userData, `dma_handle_t` \*txHandle, `dma_handle_t` \*rxHandle)  
*Initializes the FlexIO SPI slave DMA handle.*
- `status_t FLEXIO_SPI_SlaveTransferDMA` (`FLEXIO_SPI_Type` \*base, `flexio_spi_slave_dma_handle_t` \*handle, `flexio_spi_transfer_t` \*xfer)  
*Performs a non-blocking FlexIO SPI transfer using DMA.*
- `static void FLEXIO_SPI_SlaveTransferAbortDMA` (`FLEXIO_SPI_Type` \*base, `flexio_spi_slave_dma_handle_t` \*handle)  
*Aborts a FlexIO SPI transfer using DMA.*

- static status\_t **FLEXIO\_SPI\_SlaveTransferGetCountDMA** (FLEXIO\_SPI\_Type \*base, flexio\_spi\_slave\_dma\_handle\_t \*handle, size\_t \*count)  
*Gets the remaining bytes to be transferred for FlexIO SPI DMA.*

## 15.6.9.2 Data Structure Documentation

### 15.6.9.2.1 struct \_flexio\_spi\_master\_dma\_handle

typedef for flexio\_spi\_master\_dma\_handle\_t in advance.

#### Data Fields

- size\_t **transferSize**  
*Total bytes to be transferred.*
- bool **txInProgress**  
*Send transfer in progress.*
- bool **rxInProgress**  
*Receive transfer in progress.*
- dma\_handle\_t \* **txHandle**  
*DMA handler for SPI send.*
- dma\_handle\_t \* **rxHandle**  
*DMA handler for SPI receive.*
- flexio\_spi\_master\_dma\_transfer\_callback\_t **callback**  
*Callback for SPI DMA transfer.*
- void \* **userData**  
*User Data for SPI DMA callback.*

#### 15.6.9.2.1.1 Field Documentation

##### 15.6.9.2.1.1.1 size\_t flexio\_spi\_master\_dma\_handle\_t::transferSize

### 15.6.9.3 Typedef Documentation

#### 15.6.9.3.1 typedef flexio\_spi\_master\_dma\_handle\_t flexio\_spi\_slave\_dma\_handle\_t

### 15.6.9.4 Function Documentation

#### 15.6.9.4.1 status\_t FLEXIO\_SPI\_MasterTransferCreateHandleDMA ( FLEXIO\_SPI\_Type \* *base*, flexio\_spi\_master\_dma\_handle\_t \* *handle*, flexio\_spi\_master\_dma\_transfer\_callback\_t *callback*, void \* *userData*, dma\_handle\_t \* *txHandle*, dma\_handle\_t \* *rxHandle* )

This function initializes the FLEXO SPI master DMA handle which can be used for other FLEXO SPI master transactional APIs. Usually, for a specified FLEXO SPI instance, user need only call this API once to get the initialized handle.

## FlexIO SPI Driver

Parameters

|                 |                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>     | pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                         |
| <i>handle</i>   | pointer to <code>flexio_spi_master_dma_handle_t</code> structure to store the transfer state. |
| <i>callback</i> | SPI callback, NULL means no callback.                                                         |
| <i>userData</i> | callback function parameter.                                                                  |
| <i>txHandle</i> | User requested DMA handle for FlexIO SPI RX DMA transfer.                                     |
| <i>rxHandle</i> | User requested DMA handle for FlexIO SPI TX DMA transfer.                                     |

Return values

|                           |                                                    |
|---------------------------|----------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                    |
| <i>kStatus_OutOfRange</i> | The FlexIO SPI DMA type/handle table out of range. |

**15.6.9.4.2 status\_t FLEXIO\_SPI\_MasterTransferDMA ( `FLEXIO_SPI_Type * base,`  
`flexio_spi_master_dma_handle_t * handle, flexio_spi_transfer_t * xfer` )**

Note

This interface returned immediately after transfer initiates, users could call `FLEXIO_SPI_MasterGetTransferCountDMA` to poll the transfer status to check whether FlexIO SPI transfer finished.

Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                         |
| <i>handle</i> | pointer to <code>flexio_spi_master_dma_handle_t</code> structure to store the transfer state. |
| <i>xfer</i>   | Pointer to FlexIO SPI transfer structure.                                                     |

Return values

|                                |                                                      |
|--------------------------------|------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                       |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                           |
| <i>kStatus_FLEXIO_SPI_Busy</i> | FlexIO SPI is not idle, is running another transfer. |

**15.6.9.4.3 void FLEXIO\_SPI\_MasterTransferAbortDMA ( `FLEXIO_SPI_Type * base,`  
`flexio_spi_master_dma_handle_t * handle` )**

Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>handle</i> | FlexIO SPI DMA handle pointer.                        |

#### 15.6.9.4.4 **status\_t FLEXIO\_SPI\_MasterTransferGetCountDMA ( [FLEXIO\\_SPI\\_Type](#) \* *base*, [flexio\\_spi\\_master\\_dma\\_handle\\_t](#) \* *handle*, [size\\_t](#) \* *count* )**

Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_SPI_Type</a> structure.               |
| <i>handle</i> | FlexIO SPI DMA handle pointer.                                      |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

#### 15.6.9.4.5 **static void FLEXIO\_SPI\_SlaveTransferCreateHandleDMA ( [FLEXIO\\_SPI\\_Type](#) \* *base*, [flexio\\_spi\\_slave\\_dma\\_handle\\_t](#) \* *handle*, [flexio\\_spi\\_slave\\_dma\\_transfer\\_callback\\_t](#) *callback*, [void](#) \* *userData*, [dma\\_handle\\_t](#) \* *txHandle*, [dma\\_handle\\_t](#) \* *rxHandle* ) [inline], [**static**]**

This function initializes the FlexIO SPI slave DMA handle.

Parameters

|                 |                                                                                                 |
|-----------------|-------------------------------------------------------------------------------------------------|
| <i>base</i>     | pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                           |
| <i>handle</i>   | pointer to <a href="#">flexio_spi_slave_dma_handle_t</a> structure to store the transfer state. |
| <i>callback</i> | SPI callback, NULL means no callback.                                                           |
| <i>userData</i> | callback function parameter.                                                                    |
| <i>txHandle</i> | User requested DMA handle for FlexIO SPI TX DMA transfer.                                       |
| <i>rxHandle</i> | User requested DMA handle for FlexIO SPI RX DMA transfer.                                       |

#### 15.6.9.4.6 **status\_t FLEXIO\_SPI\_SlaveTransferDMA ( [FLEXIO\\_SPI\\_Type](#) \* *base*, [flexio\\_spi\\_slave\\_dma\\_handle\\_t](#) \* *handle*, [flexio\\_spi\\_transfer\\_t](#) \* *xfer* )**

Note

This interface returned immediately after transfer initiates, users could call [FLEXIO\\_SPI\\_SlaveGetTransferCountDMA](#) to poll the transfer status to check whether FlexIO SPI transfer finished.

## FlexIO SPI Driver

Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                        |
| <i>handle</i> | pointer to <code>flexio_spi_slave_dma_handle_t</code> structure to store the transfer state. |
| <i>xfer</i>   | Pointer to FlexIO SPI transfer structure.                                                    |

Return values

|                                |                                                      |
|--------------------------------|------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                       |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                           |
| <i>kStatus_FLEXIO_SPI_Busy</i> | FlexIO SPI is not idle, is running another transfer. |

**15.6.9.4.7 static void FLEXIO\_SPI\_SlaveTransferAbortDMA ( [FLEXIO\\_SPI\\_Type](#) \* *base*, `flexio_spi_slave_dma_handle_t` \* *handle* ) [inline], [static]**

Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                        |
| <i>handle</i> | pointer to <code>flexio_spi_slave_dma_handle_t</code> structure to store the transfer state. |

**15.6.9.4.8 static status\_t FLEXIO\_SPI\_SlaveTransferGetCountDMA ( [FLEXIO\\_SPI\\_Type](#) \* *base*, `flexio_spi_slave_dma_handle_t` \* *handle*, `size_t` \* *count* ) [inline], [static]**

Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | pointer to <a href="#">FLEXIO_SPI_Type</a> structure.               |
| <i>handle</i> | FlexIO SPI DMA handle pointer.                                      |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

## 15.7 FlexIO UART Driver

### 15.7.1 Overview

The KSDK provides a peripheral driver for the Universal Asynchronous Receiver/Transmitter (UART) function using the Flexible I/O.

FlexIO UART driver includes 2 parts: functional APIs and transactional APIs. Functional APIs are feature/property target low level APIs. Functional APIs can be used for the FlexIO UART initialization/configuration/operation for optimization/customization purpose. Using the functional APIs requires the knowledge of the FlexIO UART peripheral and how to organize functional APIs to meet the application requirements. All functional API use the [FLEXIO\\_UART\\_Type](#) \* as the first parameter. FlexIO UART functional operation groups provide the functional APIs set.

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs can satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the flexio\_uart\_handle\_t as the second parameter. Initialize the handle by calling the [FLEXIO\\_UART\\_TransferCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [FLEXIO\\_UART\\_SendNonBlocking\(\)](#) and [FLEXIO\\_UART\\_ReceiveNonBlocking\(\)](#) set up an interrupt for data transfer. When the transfer is complete, the upper layer is notified through a callback function with the [kStatus\\_FLEXIO\\_UART\\_TxIdle](#) and [kStatus\\_FLEXIO\\_UART\\_RxIdle](#) status.

Transactional receive APIs support the ring buffer. Prepare the memory for the ring buffer and pass in the start address and size through calling the [FLEXIO\\_UART\\_InstallRingBuffer\(\)](#). When the ring buffer is enabled, the received data is saved to the ring buffer in the background. The function [FLEXIO\\_UART\\_ReceiveNonBlocking\(\)](#) first gets data from the ring buffer. If ring buffer does not have enough data, the function returns the data to the ring buffer and saves the received data to user memory. When all data is received, the upper layer is informed through a callback with the [status\\_kStatus\\_FLEXIO\\_UART\\_RxIdle](#) status.

If the receive ring buffer is full, the upper layer is informed through a callback with status [kStatus\\_FLEXIO\\_UART\\_RxRingBufferOverrun](#). In the callback function, the upper layer reads data from the ring buffer. If not, the oldest data is overwritten by the new data.

The ring buffer size is specified when calling the [FLEXIO\\_UART\\_InstallRingBuffer](#). Note that one byte is reserved for the ring buffer maintenance. Create a handle as follows:

```
FLEXIO_UART_InstallRingBuffer(&uartDev, &handle, &ringBuffer, 32);
```

In this example, the buffer size is 32. However, only 31 bytes are used for saving data.

### 15.7.2 Typical use case

#### 15.7.2.1 FlexIO UART send/receive using a polling method

```
uint8_t ch;
```

## FlexIO UART Driver

```
FLEXIO_UART_Type uartDev;
flexio_uart_user_config user_config;
FLEXIO_UART_GetDefaultConfig(&user_config);
user_config.baudRate_Bps = 115200U;
user_config.enableUart = true;

uartDev.flexioBase = BOARD_FLEXIO_BASE;
uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
uartDev.shifterIndex[0] = 0U;
uartDev.shifterIndex[1] = 1U;
uartDev.timerIndex[0] = 0U;
uartDev.timerIndex[1] = 1U;

FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);

FLEXIO_UART_WriteBlocking(&uartDev, txbuff, sizeof(txbuff));

while(1)
{
    FLEXIO_UART_ReadBlocking(&uartDev, &ch, 1);
    FLEXIO_UART_WriteBlocking(&uartDev, &ch, 1);
}
```

### 15.7.2.2 FlexIO UART send/receive using an interrupt method

```
FLEXIO_UART_Type uartDev;
flexio_uart_handle_t g_uartHandle;
flexio_uart_config_t user_config;
flexio_uart_transfer_t sendXfer;
flexio_uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = {'H', 'e', 'l', 'l', 'o'};
uint8_t receiveData[32];

void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
                             status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_UART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_FLEXIO_UART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    FLEXIO_UART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableUart = true;

    uartDev.flexioBase = BOARD_FLEXIO_BASE;
    uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
    uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
    uartDev.shifterIndex[0] = 0U;
    uartDev.shifterIndex[1] = 1U;
```

```

uartDev.timerIndex[0] = 0U;
uartDev.timerIndex[1] = 1U;

FLEXIO_UART_Init(&uartDev, &user_config, 120000000U);
FLEXIO_UART_TransferCreateHandle(&uartDev, &g_uartHandle,
    FLEXIO_UART_UserCallback, NULL);

// Prepares to send.
sendXfer.data = sendData;
sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
txFinished = false;

// Sends out.
FLEXIO_UART_SendNonBlocking(&uartDev, &g_uartHandle, &sendXfer);

// Send finished.
while (!txFinished)
{
}

// Prepares to receive.
receiveXfer.data = receiveData;
receiveXfer.dataSize = sizeof(receiveData)/sizeof(receiveData[0]);
rxFinished = false;

// Receives.
FLEXIO_UART_ReceiveNonBlocking(&uartDev, &g_uartHandle, &receiveXfer, NULL);

// Receive finished.
while (!rxFinished)
{
}

// ...
}

```

### 15.7.2.3 FlexIO UART receive using the ringbuffer feature

```

#define RING_BUFFER_SIZE 64
#define RX_DATA_SIZE     32

FLEXIO_UART_Type uartDev;
flexio_uart_handle_t g_uartHandle;
flexio_uart_config_t user_config;
flexio_uart_transfer_t sendXfer;
flexio_uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t receiveData[RX_DATA_SIZE];
uint8_t ringBuffer[RING_BUFFER_SIZE];

void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
    status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_UART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    size_t bytesRead;

```

## FlexIO UART Driver

```
//...

FLEXIO_UART_GetDefaultConfig(&user_config);
user_config.baudRate_Bps = 115200U;
user_config.enableUart = true;

uartDev.flexioBase = BOARD_FLEXIO_BASE;
uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
uartDev.shifterIndex[0] = 0U;
uartDev.shifterIndex[1] = 1U;
uartDev.timerIndex[0] = 0U;
uartDev.timerIndex[1] = 1U;

FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);
FLEXIO_UART_TransferCreateHandle(&uartDev, &g_uartHandle,
    FLEXIO_UART_UserCallback, NULL);
FLEXIO_UART_InstallRingBuffer(&uartDev, &g_uartHandle, ringBuffer, RING_BUFFER_SIZE);

// Receive is working in the background to the ring buffer.

// Prepares to receive.
receiveXfer.data = receiveData;
receiveXfer.dataSize = RX_DATA_SIZE;
rxFinished = false;

// Receives.
FLEXIO_UART_ReceiveNonBlocking(&uartDev, &g_uartHandle, &receiveXfer, &bytesRead);

if (bytesRead == RX_DATA_SIZE) /* Have read enough data. */
{
    ;
}
else
{
    if (bytesRead) /* Received some data, process first. */
    {
        ;
    }

    // Receive finished.
    while (!rxFinished)
    {
    }
}

// ...
}
```

### 15.7.2.4 FlexIO UART send/receive using a DMA method

```
FLEXIO_UART_Type uartDev;
flexio_uart_handle_t g_uartHandle;
dma_handle_t g_uartTxDmaHandle;
dma_handle_t g_uartRxDmaHandle;
flexio_uart_config_t user_config;
flexio_uart_transfer_t sendXfer;
flexio_uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = {'H', 'e', 'l', 'l', 'o'};
uint8_t receiveData[32];

void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
    status_t status, void *userData)
```

```

{
    userData = userData;

    if (kStatus_FLEXIO_UART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_FLEXIO_UART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    FLEXIO_UART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableUart = true;

    uartDev.flexioBase = BOARD_FLEXIO_BASE;
    uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
    uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
    uartDev.shifterIndex[0] = 0U;
    uartDev.shifterIndex[1] = 1U;
    uartDev.timerIndex[0] = 0U;
    uartDev.timerIndex[1] = 1U;
    FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);

    /*Initializes the DMA for the example*/
    DMAMGR_Init();

    dma_request_source_tx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + uartDev.
        shifterIndex[0]);
    dma_request_source_rx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + uartDev.
        shifterIndex[1]);

    /* Requests DMA channels for transmit and receive. */
    DMAMGR_RequestChannel((dma_request_source_t)dma_request_source_tx, 0, &
        g_uartTxDmaHandle);
    DMAMGR_RequestChannel((dma_request_source_t)dma_request_source_rx, 1, &
        g_uartRxDmaHandle);

    FLEXIO_UART_TransferCreateHandleDMA(&uartDev, &g_uartHandle,
        FLEXIO_UART_UserCallback, NULL, &g_uartTxDmaHandle, &g_uartRxDmaHandle);

    // Prepares to send.
    sendXfer.data = sendData;
    sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
    txFinished = false;

    // Sends out.
    FLEXIO_UART_SendDMA(&uartDev, &g_uartHandle, &sendXfer);

    // Send finished.
    while (!txFinished)
    {

        // Prepares to receive.
        receiveXfer.data = receiveData;
        receiveXfer.dataSize = sizeof(receiveData)/sizeof(receiveData[0]);
        rxFinished = false;

        // Receives.
        FLEXIO_UART_ReceiveDMA(&uartDev, &g_uartHandle, &receiveXfer, NULL);
}

```

## FlexIO UART Driver

```
// Receive finished.  
while (!rxFinished)  
{  
}  
  
// ...  
}
```

## Modules

- FlexIO DMA UART Driver
- FlexIO eDMA UART Driver

## Data Structures

- struct [FLEXIO\\_UART\\_Type](#)  
*Define FlexIO UART access structure typedef.* [More...](#)
- struct [flexio\\_uart\\_config\\_t](#)  
*Define FlexIO UART user configuration structure.* [More...](#)
- struct [flexio\\_uart\\_transfer\\_t](#)  
*Define FlexIO UART transfer structure.* [More...](#)
- struct [flexio\\_uart\\_handle\\_t](#)  
*Define FLEXIO UART handle structure.* [More...](#)

## Typedefs

- [typedef void\(\\* flexio\\_uart\\_transfer\\_callback\\_t \)\(FLEXIO\\_UART\\_Type \\*base, flexio\\_uart\\_handle\\_t \\*handle, status\\_t status, void \\*userData\)](#)  
*FlexIO UART transfer callback function.*

## Enumerations

- enum [\\_flexio\\_uart\\_status](#) {  
    kStatus\_FLEXIO\_UART\_TxBusy = MAKE\_STATUS(kStatusGroup\_FLEXIO\_UART, 0),  
    kStatus\_FLEXIO\_UART\_RxBusy = MAKE\_STATUS(kStatusGroup\_FLEXIO\_UART, 1),  
    kStatus\_FLEXIO\_UART\_TxIdle = MAKE\_STATUS(kStatusGroup\_FLEXIO\_UART, 2),  
    kStatus\_FLEXIO\_UART\_RxIdle = MAKE\_STATUS(kStatusGroup\_FLEXIO\_UART, 3),  
    kStatus\_FLEXIO\_UART\_ERROR = MAKE\_STATUS(kStatusGroup\_FLEXIO\_UART, 4),  
    kStatus\_FLEXIO\_UART\_RxRingBufferOverrun,  
    kStatus\_FLEXIO\_UART\_RxHardwareOverrun = MAKE\_STATUS(kStatusGroup\_FLEXIO\_UART, 6) }  
*Error codes for the UART driver.*
- enum [flexio\\_uart\\_bit\\_count\\_per\\_char\\_t](#) {  
    kFLEXIO\_UART\_7BitsPerChar = 7U,  
    kFLEXIO\_UART\_8BitsPerChar = 8U,

- ```

kFLEXIO_UART_9BitsPerChar = 9U }

FlexIO UART bit count per char.
• enum _flexio_uart_interrupt_enable {
    kFLEXIO_UART_TxDataRegEmptyInterruptEnable = 0x1U,
    kFLEXIO_UART_RxDataRegFullInterruptEnable = 0x2U }

Define FlexIO UART interrupt mask.
• enum _flexio_uart_status_flags {
    kFLEXIO_UART_TxDataRegEmptyFlag = 0x1U,
    kFLEXIO_UART_RxDataRegFullFlag = 0x2U,
    kFLEXIO_UART_RxOverRunFlag = 0x4U }

Define FlexIO UART status mask.

```

## Driver version

- #define **FSL\_FLEXIO\_UART\_DRIVER\_VERSION** (MAKE\_VERSION(2, 1, 1))  
*FlexIO UART driver version 2.1.1.*

## Initialization and deinitialization

- void **FLEXIO\_UART\_Init** (**FLEXIO\_UART\_Type** \*base, const **flexio\_uart\_config\_t** \*userConfig, uint32\_t srcClock\_Hz)  
*Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration.*
- void **FLEXIO\_UART\_Deinit** (**FLEXIO\_UART\_Type** \*base)  
*Disables the FlexIO UART and gates the FlexIO clock.*
- void **FLEXIO\_UART\_GetDefaultConfig** (**flexio\_uart\_config\_t** \*userConfig)  
*Gets the default configuration to configure the FlexIO UART.*

## Status

- uint32\_t **FLEXIO\_UART\_GetStatusFlags** (**FLEXIO\_UART\_Type** \*base)  
*Gets the FlexIO UART status flags.*
- void **FLEXIO\_UART\_ClearStatusFlags** (**FLEXIO\_UART\_Type** \*base, uint32\_t mask)  
*Gets the FlexIO UART status flags.*

## Interrupts

- void **FLEXIO\_UART\_EnableInterrupts** (**FLEXIO\_UART\_Type** \*base, uint32\_t mask)  
*Enables the FlexIO UART interrupt.*
- void **FLEXIO\_UART\_DisableInterrupts** (**FLEXIO\_UART\_Type** \*base, uint32\_t mask)  
*Disables the FlexIO UART interrupt.*

## FlexIO UART Driver

### DMA Control

- static uint32\_t **FLEXIO\_UART\_GetTxDataRegisterAddress** (FLEXIO\_UART\_Type \*base)  
*Gets the FlexIO UART transmit data register address.*
- static uint32\_t **FLEXIO\_UART\_GetRxDataRegisterAddress** (FLEXIO\_UART\_Type \*base)  
*Gets the FlexIO UART receive data register address.*
- static void **FLEXIO\_UART\_EnableTxDMA** (FLEXIO\_UART\_Type \*base, bool enable)  
*Enables/disables the FlexIO UART transmit DMA.*
- static void **FLEXIO\_UART\_EnableRxDMA** (FLEXIO\_UART\_Type \*base, bool enable)  
*Enables/disables the FlexIO UART receive DMA.*

### Bus Operations

- static void **FLEXIO\_UART\_Enable** (FLEXIO\_UART\_Type \*base, bool enable)  
*Enables/disables the FlexIO UART module operation.*
- static void **FLEXIO\_UART\_WriteByte** (FLEXIO\_UART\_Type \*base, const uint8\_t \*buffer)  
*Writes one byte of data.*
- static void **FLEXIO\_UART\_ReadByte** (FLEXIO\_UART\_Type \*base, uint8\_t \*buffer)  
*Reads one byte of data.*
- void **FLEXIO\_UART\_WriteBlocking** (FLEXIO\_UART\_Type \*base, const uint8\_t \*txData, size\_t txSize)  
*Sends a buffer of data bytes.*
- void **FLEXIO\_UART\_ReadBlocking** (FLEXIO\_UART\_Type \*base, uint8\_t \*rxData, size\_t rxSize)  
*Receives a buffer of bytes.*

### Transactional

- status\_t **FLEXIO\_UART\_TransferCreateHandle** (FLEXIO\_UART\_Type \*base, flexio\_uart\_handle\_t \*handle, flexio\_uart\_transfer\_callback\_t callback, void \*userData)  
*Initializes the UART handle.*
- void **FLEXIO\_UART\_TransferStartRingBuffer** (FLEXIO\_UART\_Type \*base, flexio\_uart\_handle\_t \*handle, uint8\_t \*ringBuffer, size\_t ringBufferSize)  
*Sets up the RX ring buffer.*
- void **FLEXIO\_UART\_TransferStopRingBuffer** (FLEXIO\_UART\_Type \*base, flexio\_uart\_handle\_t \*handle)  
*Aborts the background transfer and uninstalls the ring buffer.*
- status\_t **FLEXIO\_UART\_TransferSendNonBlocking** (FLEXIO\_UART\_Type \*base, flexio\_uart\_handle\_t \*handle, flexio\_uart\_transfer\_t \*xfer)  
*Transmits a buffer of data using the interrupt method.*
- void **FLEXIO\_UART\_TransferAbortSend** (FLEXIO\_UART\_Type \*base, flexio\_uart\_handle\_t \*handle)  
*Aborts the interrupt-driven data transmit.*
- status\_t **FLEXIO\_UART\_TransferGetSendCount** (FLEXIO\_UART\_Type \*base, flexio\_uart\_handle\_t \*handle, size\_t \*count)  
*Gets the number of remaining bytes not sent.*

- status\_t [FLEXIO\\_UART\\_TransferReceiveNonBlocking](#) ([FLEXIO\\_UART\\_Type](#) \*base, [flexio\\_uart\\_handle\\_t](#) \*handle, [flexio\\_uart\\_transfer\\_t](#) \*xfer, [size\\_t](#) \*receivedBytes)  
*Receives a buffer of data using the interrupt method.*
- void [FLEXIO\\_UART\\_TransferAbortReceive](#) ([FLEXIO\\_UART\\_Type](#) \*base, [flexio\\_uart\\_handle\\_t](#) \*handle)  
*Aborts the receive data which was using IRQ.*
- status\_t [FLEXIO\\_UART\\_TransferGetReceiveCount](#) ([FLEXIO\\_UART\\_Type](#) \*base, [flexio\\_uart\\_handle\\_t](#) \*handle, [size\\_t](#) \*count)  
*Gets the number of remaining bytes not received.*
- void [FLEXIO\\_UART\\_TransferHandleIRQ](#) (void \*uartType, void \*uartHandle)  
*FlexIO UART IRQ handler function.*

### 15.7.3 Data Structure Documentation

#### 15.7.3.1 struct [FLEXIO\\_UART\\_Type](#)

##### Data Fields

- [FLEXIO\\_Type](#) \* [flexioBase](#)  
*FlexIO base pointer.*
- [uint8\\_t](#) [TxPinIndex](#)  
*Pin select for UART\_Tx.*
- [uint8\\_t](#) [RxPinIndex](#)  
*Pin select for UART\_Rx.*
- [uint8\\_t](#) [shifterIndex](#) [2]  
*Shifter index used in FlexIO UART.*
- [uint8\\_t](#) [timerIndex](#) [2]  
*Timer index used in FlexIO UART.*

##### 15.7.3.1.0.1 Field Documentation

###### 15.7.3.1.0.1.1 [FLEXIO\\_Type\\*](#) [FLEXIO\\_UART\\_Type::flexioBase](#)

###### 15.7.3.1.0.1.2 [uint8\\_t](#) [FLEXIO\\_UART\\_Type::TxPinIndex](#)

###### 15.7.3.1.0.1.3 [uint8\\_t](#) [FLEXIO\\_UART\\_Type::RxPinIndex](#)

###### 15.7.3.1.0.1.4 [uint8\\_t](#) [FLEXIO\\_UART\\_Type::shifterIndex\[2\]](#)

###### 15.7.3.1.0.1.5 [uint8\\_t](#) [FLEXIO\\_UART\\_Type::timerIndex\[2\]](#)

#### 15.7.3.2 struct [flexio\\_uart\\_config\\_t](#)

##### Data Fields

- bool [enableUart](#)  
*Enable/disable FlexIO UART TX & RX.*
- bool [enableInDoze](#)  
*Enable/disable FlexIO operation in doze mode.*

## FlexIO UART Driver

- bool `enableInDebug`  
*Enable/disable FlexIO operation in debug mode.*
- bool `enableFastAccess`  
*Enable/disable fast access to FlexIO registers,  
fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- uint32\_t `baudRate_Bps`  
*Baud rate in Bps.*
- `flexio_uart_bit_count_per_char_t bitCountPerChar`  
*number of bits, 7/8/9 -bit*

### 15.7.3.2.0.2 Field Documentation

#### 15.7.3.2.0.2.1 bool `flexio_uart_config_t::enableUart`

#### 15.7.3.2.0.2.2 bool `flexio_uart_config_t::enableFastAccess`

#### 15.7.3.2.0.2.3 uint32\_t `flexio_uart_config_t::baudRate_Bps`

### 15.7.3.3 struct `flexio_uart_transfer_t`

#### Data Fields

- uint8\_t \* `data`  
*Transfer buffer.*
- size\_t `dataSize`  
*Transfer size.*

### 15.7.3.4 struct `_flexio_uart_handle`

#### Data Fields

- uint8\_t \*volatile `txData`  
*Address of remaining data to send.*
- volatile size\_t `txDataSize`  
*Size of the remaining data to send.*
- uint8\_t \*volatile `rxData`  
*Address of remaining data to receive.*
- volatile size\_t `rxDataSize`  
*Size of the remaining data to receive.*
- size\_t `txSize`  
*Total bytes to be sent.*
- size\_t `rxSize`  
*Total bytes to be received.*
- uint8\_t \* `rxRingBuffer`  
*Start address of the receiver ring buffer.*
- size\_t `rxRingBufferSize`  
*Size of the ring buffer.*
- volatile uint16\_t `rxRingBufferHead`  
*Index for the driver to store received data into ring buffer.*
- volatile uint16\_t `rxRingBufferTail`

*Index for the user to get data from the ring buffer.*

- **flexio\_uart\_transfer\_callback\_t callback**

*Callback function.*

- void \* **userData**

*UART callback function parameter.*

- volatile uint8\_t **txState**

*TX transfer state.*

- volatile uint8\_t **rxState**

*RX transfer state.*

## FlexIO UART Driver

### 15.7.3.4.0.3 Field Documentation

- 15.7.3.4.0.3.1 `uint8_t* volatile flexio_uart_handle_t::txData`
- 15.7.3.4.0.3.2 `volatile size_t flexio_uart_handle_t::txDataSize`
- 15.7.3.4.0.3.3 `uint8_t* volatile flexio_uart_handle_t::rxData`
- 15.7.3.4.0.3.4 `volatile size_t flexio_uart_handle_t::rxDataSize`
- 15.7.3.4.0.3.5 `size_t flexio_uart_handle_t::txSize`
- 15.7.3.4.0.3.6 `size_t flexio_uart_handle_t::rxSize`
- 15.7.3.4.0.3.7 `uint8_t* flexio_uart_handle_t::rxRingBuffer`
- 15.7.3.4.0.3.8 `size_t flexio_uart_handle_t::rxRingBufferSize`
- 15.7.3.4.0.3.9 `volatile uint16_t flexio_uart_handle_t::rxRingBufferHead`
- 15.7.3.4.0.3.10 `volatile uint16_t flexio_uart_handle_t::rxRingBufferTail`
- 15.7.3.4.0.3.11 `flexio_uart_transfer_callback_t flexio_uart_handle_t::callback`
- 15.7.3.4.0.3.12 `void* flexio_uart_handle_t::userData`
- 15.7.3.4.0.3.13 `volatile uint8_t flexio_uart_handle_t::txState`

### 15.7.4 Macro Definition Documentation

- 15.7.4.1 `#define FSL_FLEXIO_UART_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`

### 15.7.5 Typedef Documentation

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

### 15.7.6 Enumeration Type Documentation

#### 15.7.6.1 enum \_flexio\_uart\_status

Enumerator

- `kStatus_FLEXIO_UART_TxBusy` Transmitter is busy.
- `kStatus_FLEXIO_UART_RxBusy` Receiver is busy.
- `kStatus_FLEXIO_UART_TxIdle` UART transmitter is idle.
- `kStatus_FLEXIO_UART_RxIdle` UART receiver is idle.
- `kStatus_FLEXIO_UART_ERROR` ERROR happens on UART.

*kStatus\_FLEXIO\_UART\_RxRingBufferOverrun*  UART RX software ring buffer overrun.  
*kStatus\_FLEXIO\_UART\_RxHardwareOverrun*  UART RX receiver overrun.

### 15.7.6.2 enum flexio\_uart\_bit\_count\_per\_char\_t

Enumerator

*kFLEXIO\_UART\_7BitsPerChar*  7-bit data characters  
*kFLEXIO\_UART\_8BitsPerChar*  8-bit data characters  
*kFLEXIO\_UART\_9BitsPerChar*  9-bit data characters

### 15.7.6.3 enum \_flexio\_uart\_interrupt\_enable

Enumerator

*kFLEXIO\_UART\_TxDataRegEmptyInterruptEnable*  Transmit buffer empty interrupt enable.  
*kFLEXIO\_UART\_RxDataRegFullInterruptEnable*  Receive buffer full interrupt enable.

### 15.7.6.4 enum \_flexio\_uart\_status\_flags

Enumerator

*kFLEXIO\_UART\_TxDataRegEmptyFlag*  Transmit buffer empty flag.  
*kFLEXIO\_UART\_RxDataRegFullFlag*  Receive buffer full flag.  
*kFLEXIO\_UART\_RxOverRunFlag*  Receive buffer over run flag.

## 15.7.7 Function Documentation

### 15.7.7.1 void FLEXIO\_UART\_Init ( FLEXIO\_UART\_Type \* *base*, const flexio\_uart\_config\_t \* *userConfig*, uint32\_t *srcClock\_Hz* )

The configuration structure can be filled by the user, or be set with default values by [FLEXIO\\_UART\\_GetDefaultConfig\(\)](#).

Example

```
FLEXIO_UART_Type base = {
    .flexioBase = FLEXIO,
    .TxPinIndex = 0,
    .RxPinIndex = 1,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_uart_config_t config = {
    .enableInDoze = false,
```

## FlexIO UART Driver

```
.enableInDebug = true,  
.enableFastAccess = false,  
.baudRate_Bps = 115200U,  
.bitCountPerChar = 8  
};  
FLEXIO_UART_Init(base, &config, srcClock_Hz);
```

Parameters

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

### 15.7.7.2 void **FLEXIO\_UART\_Deinit**( **FLEXIO\_UART\_Type** \* *base* )

Note

After calling this API, call the **FLEXIO\_UART\_Init** to use the FlexIO UART module.

Parameters

<i>base</i>	pointer to <b>FLEXIO_UART_Type</b> structure
-------------	--

### 15.7.7.3 void **FLEXIO\_UART\_GetDefaultConfig**( **flexio\_uart\_config\_t** \* *userConfig* )

The configuration can be used directly for calling the **FLEXIO\_UART\_Init()**. Example:

```
flexio_uart_config_t config;  
FLEXIO_UART_GetDefaultConfig(&userConfig);
```

Parameters

<i>userConfig</i>	Pointer to the <b>flexio_uart_config_t</b> structure.
-------------------	---

### 15.7.7.4 uint32\_t **FLEXIO\_UART\_GetStatusFlags**( **FLEXIO\_UART\_Type** \* *base* )

Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
-------------	--

Returns

FlexIO UART status flags.

#### 15.7.7.5 void FLEXIO\_UART\_ClearStatusFlags ( [FLEXIO\\_UART\\_Type](#) \* *base*, [uint32\\_t](#) *mask* )

Parameters

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

#### 15.7.7.6 void FLEXIO\_UART\_EnableInterrupts ( [FLEXIO\\_UART\\_Type](#) \* *base*, [uint32\\_t](#) *mask* )

This function enables the FlexIO UART interrupt.

Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
<i>mask</i>	Interrupt source.

#### 15.7.7.7 void FLEXIO\_UART\_DisableInterrupts ( [FLEXIO\\_UART\\_Type](#) \* *base*, [uint32\\_t](#) *mask* )

This function disables the FlexIO UART interrupt.

Parameters

## FlexIO UART Driver

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
<i>mask</i>	Interrupt source.

### **15.7.7.8 static uint32\_t FLEXIO\_UART\_GetTxDataRegisterAddress ( FLEXIO\_UART\_Type \* *base* ) [inline], [static]**

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
-------------	--

Returns

FlexIO UART transmit data register address.

### **15.7.7.9 static uint32\_t FLEXIO\_UART\_GetRxDataRegisterAddress ( FLEXIO\_UART\_Type \* *base* ) [inline], [static]**

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
-------------	--

Returns

FlexIO UART receive data register address.

### **15.7.7.10 static void FLEXIO\_UART\_EnableTxDMA ( FLEXIO\_UART\_Type \* *base*, bool *enable* ) [inline], [static]**

This function enables/disables the FlexIO UART Tx DMA, which means asserting the kFLEXIO\_UART\_TxDataRegEmptyFlag does/doesn't trigger the DMA request.

Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
<i>enable</i>	True to enable, false to disable.

#### 15.7.7.11 static void FLEXIO\_UART\_EnableRxDMA ( [FLEXIO\\_UART\\_Type](#) \* *base*, bool *enable* ) [inline], [static]

This function enables/disables the FlexIO UART Rx DMA, which means asserting kFLEXIO\_UART\_RxDataRegFullFlag does/doesn't trigger the DMA request.

Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
<i>enable</i>	True to enable, false to disable.

#### 15.7.7.12 static void FLEXIO\_UART\_Enable ( [FLEXIO\\_UART\\_Type](#) \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> .
<i>enable</i>	True to enable, false to disable.

#### 15.7.7.13 static void FLEXIO\_UART\_WriteByte ( [FLEXIO\\_UART\\_Type](#) \* *base*, const uint8\_t \* *buffer* ) [inline], [static]

Note

This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
<i>buffer</i>	The data bytes to send.

#### 15.7.7.14 static void FLEXIO\_UART\_ReadByte ( [FLEXIO\\_UART\\_Type](#) \* *base*, uint8\_t \* *buffer* ) [inline], [static]

## FlexIO UART Driver

### Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

### Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
<i>buffer</i>	The buffer to store the received bytes.

### **15.7.7.15 void FLEXIO\_UART\_WriteBlocking ( [FLEXIO\\_UART\\_Type](#) \* *base*, const [uint8\\_t](#) \* *txData*, [size\\_t](#) *txSize* )**

### Note

This function blocks using the polling method until all bytes have been sent.

### Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
<i>txData</i>	The data bytes to send.
<i>txSize</i>	The number of data bytes to send.

### **15.7.7.16 void FLEXIO\_UART\_ReadBlocking ( [FLEXIO\\_UART\\_Type](#) \* *base*, [uint8\\_t](#) \* *rxData*, [size\\_t](#) *rxSize* )**

### Note

This function blocks using the polling method until all bytes have been received.

### Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
<i>rxData</i>	The buffer to store the received bytes.
<i>rxSize</i>	The number of data bytes to be received.

### **15.7.7.17 [status\\_t](#) FLEXIO\_UART\_TransferCreateHandle ( [FLEXIO\\_UART\\_Type](#) \* *base*, [flexio\\_uart\\_handle\\_t](#) \* *handle*, [flexio\\_uart\\_transfer\\_callback\\_t](#) *callback*, [void](#) \* *userData* )**

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the "background" receiving, which means that user can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the [FLEXIO\\_UART\\_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, 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>	to <a href="#">FLEXIO_UART_Type</a> structure.
<i>handle</i>	Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state.
<i>callback</i>	The callback function.
<i>userData</i>	The parameter of the callback function.

#### Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_OutOfRange</i>	The FlexIO type/handle/ISR table out of range.

### 15.7.7.18 void FLEXIO\_UART\_TransferStartRingBuffer ( [FLEXIO\\_UART\\_Type](#) \* *base*, `flexio_uart_handle_t` \* *handle*, `uint8_t` \* *ringBuffer*, `size_t` *ringBufferSize* )

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the [UART\\_ReceiveNonBlocking\(\)](#) API. If there are already data received in the ring buffer, 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>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
<i>handle</i>	Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state.
<i>ringBuffer</i>	Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
<i>ringBufferSize</i>	Size of the ring buffer.

**15.7.7.19 void FLEXIO\_UART\_TransferStopRingBuffer ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_handle\_t \* *handle* )**

This function aborts the background transfer and uninstalls the ring buffer.

## Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
<i>handle</i>	Pointer to the <a href="#">flexio_uart_handle_t</a> structure to store the transfer state.

**15.7.7.20 status\_t FLEXIO\_UART\_TransferSendNonBlocking ( [FLEXIO\\_UART\\_Type](#) \* *base*, [flexio\\_uart\\_handle\\_t](#) \* *handle*, [flexio\\_uart\\_transfer\\_t](#) \* *xfer* )**

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data are written to TX register in ISR, the FlexIO UART driver calls the callback function and passes the [kStatus\\_FLEXIO\\_UART\\_TxIdle](#) as status parameter.

## Note

The [kStatus\\_FLEXIO\\_UART\\_TxIdle](#) is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out.

## Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
<i>handle</i>	Pointer to the <a href="#">flexio_uart_handle_t</a> structure to store the transfer state.
<i>xfer</i>	FlexIO UART transfer structure. See <a href="#">flexio_uart_transfer_t</a> .

## Return values

<i>kStatus_Success</i>	Successfully starts the data transmission.
<i>kStatus_UART_TxBusy</i>	Previous transmission still not finished, data not written to the TX register.

**15.7.7.21 void FLEXIO\_UART\_TransferAbortSend ( [FLEXIO\\_UART\\_Type](#) \* *base*, [flexio\\_uart\\_handle\\_t](#) \* *handle* )**

This function aborts the interrupt-driven data sending. Get the *remainBytes* to know how many bytes are still not sent out.

## Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
<i>handle</i>	Pointer to the <a href="#">flexio_uart_handle_t</a> structure to store the transfer state.

### 15.7.7.22 `status_t FLEXIO_UART_TransferGetSendCount( FLEXIO_UART_Type * base, flexio_uart_handle_t * handle, size_t * count )`

This function gets the number of remaining bytes not sent driven by interrupt.

## Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
<i>handle</i>	Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state.
<i>count</i>	Number of bytes sent so far by the non-blocking transaction.

## Return values

<i>kStatus_InvalidArgument</i>	count is Invalid.
<i>kStatus_Success</i>	Successfully return the count.

**15.7.7.23 `status_t FLEXIO_UART_TransferReceiveNonBlocking ( FLEXIO_UART_Type * base, flexio_uart_handle_t * handle, flexio_uart_transfer_t * xfer, size_t * receivedBytes )`**

This function receives data using the interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter `kStatus_UART_RxIdle`. For example, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to `xfer->data`. This function returns with the parameter `receivedBytes` set to 5. For the last 5 bytes, newly arrived data is saved from the `xfer->data[5]`. When 5 bytes are received, the UART driver notifies upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to `xfer->data`. When all data is received, the upper layer is notified.

## Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
<i>handle</i>	Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state.
<i>xfer</i>	UART transfer structure. See <a href="#">flexio_uart_transfer_t</a> .
<i>receivedBytes</i>	Bytes received from the ring buffer directly.

## Return values

<i>kStatus_Success</i>	Successfully queue the transfer into the transmit queue.
<i>kStatus_FLEXIO_UART_RxBusy</i>	Previous receive request is not finished.

**15.7.7.24 void FLEXIO\_UART\_TransferAbortReceive ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_handle\_t \* *handle* )**

This function aborts the receive data which was using IRQ.

Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
<i>handle</i>	Pointer to the flexio_uart_handle_t structure to store the transfer state.

#### 15.7.7.25 status\_t FLEXIO\_UART\_TransferGetReceiveCount ( [FLEXIO\\_UART\\_Type](#) \* *base*, flexio\_uart\_handle\_t \* *handle*, size\_t \* *count* )

This function gets the number of remaining bytes not received driven by interrupt.

Parameters

<i>base</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
<i>handle</i>	Pointer to the flexio_uart_handle_t structure to store the transfer state.
<i>count</i>	Number of bytes received so far by the non-blocking transaction.

Return values

<i>kStatus_InvalidArgument</i>	count is Invalid.
<i>kStatus_Success</i>	Successfully return the count.

#### 15.7.7.26 void FLEXIO\_UART\_TransferHandleIRQ ( void \* *uartType*, void \* *uartHandle* )

This function processes the FlexIO UART transmit and receives the IRQ request.

Parameters

<i>uartType</i>	Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.
<i>uartHandle</i>	Pointer to the flexio_uart_handle_t structure to store the transfer state.

### 15.7.8 FlexIO eDMA UART Driver

#### 15.7.8.1 Overview

#### Data Structures

- struct `flexio_uart_edma_handle_t`  
*UART eDMA handle.* [More...](#)

#### TypeDefs

- typedef void(\* `flexio_uart_edma_transfer_callback_t`)(`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `status_t` status, `void` \*userData)  
*UART transfer callback function.*

#### eDMA transactional

- `status_t FLEXIO_UART_TransferCreateHandleEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `flexio_uart_edma_transfer_callback_t` callback, `void` \*userData, `edma_handle_t` \*txEdmaHandle, `edma_handle_t` \*rxEdmaHandle)  
*Initializes the UART handle which is used in transactional functions.*
- `status_t FLEXIO_UART_TransferSendEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `flexio_uart_transfer_t` \*xfer)  
*Sends data using eDMA.*
- `status_t FLEXIO_UART_TransferReceiveEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `flexio_uart_transfer_t` \*xfer)  
*Receives data using eDMA.*
- `void FLEXIO_UART_TransferAbortSendEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle)  
*Aborts the sent data which using eDMA.*
- `void FLEXIO_UART_TransferAbortReceiveEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle)  
*Aborts the receive data which using eDMA.*
- `status_t FLEXIO_UART_TransferGetSendCountEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `size_t` \*count)  
*Gets the number of bytes still not sent out.*
- `status_t FLEXIO_UART_TransferGetReceiveCountEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `size_t` \*count)  
*Gets the number of bytes still not received.*

### 15.7.8.2 Data Structure Documentation

#### 15.7.8.2.1 struct \_flexio\_uart\_edma\_handle

##### Data Fields

- `flexio_uart_edma_transfer_callback_t callback`  
*Callback function.*
- `void *userData`  
*UART callback function parameter.*
- `size_t txSize`  
*Total bytes to be sent.*
- `size_t rxSize`  
*Total bytes to be received.*
- `edma_handle_t *txEdmaHandle`  
*The eDMA TX channel used.*
- `edma_handle_t *rxEdmaHandle`  
*The eDMA RX channel used.*
- `volatile uint8_t txState`  
*TX transfer state.*
- `volatile uint8_t rxState`  
*RX transfer state.*

##### 15.7.8.2.1.1 Field Documentation

15.7.8.2.1.1.1 `flexio_uart_edma_transfer_callback_t flexio_uart_edma_handle_t::callback`

15.7.8.2.1.1.2 `void* flexio_uart_edma_handle_t::userData`

15.7.8.2.1.1.3 `size_t flexio_uart_edma_handle_t::txSize`

15.7.8.2.1.1.4 `size_t flexio_uart_edma_handle_t::rxSize`

15.7.8.2.1.1.5 `edma_handle_t* flexio_uart_edma_handle_t::txEdmaHandle`

15.7.8.2.1.1.6 `edma_handle_t* flexio_uart_edma_handle_t::rxEdmaHandle`

15.7.8.2.1.1.7 `volatile uint8_t flexio_uart_edma_handle_t::txState`

##### 15.7.8.3 Typedef Documentation

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

##### 15.7.8.4 Function Documentation

15.7.8.4.1 `status_t FLEXIO_UART_TransferCreateHandleEDMA ( FLEXIO_UART_Type * base, flexio_uart_edma_handle_t * handle, flexio_uart_edma_transfer_callback_t callback, void * userData, edma_handle_t * txEdmaHandle, edma_handle_t * rxEdmaHandle )`

## FlexIO UART Driver

Parameters

<i>base</i>	pointer to <a href="#">FLEXIO_UART_Type</a> .
<i>handle</i>	Pointer to flexio_uart_edma_handle_t structure.
<i>callback</i>	The callback function.
<i>userData</i>	The parameter of the callback function.
<i>rxEdmaHandle</i>	User requested DMA handle for RX DMA transfer.
<i>txEdmaHandle</i>	User requested DMA handle for TX DMA transfer.

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_OutOfRange</i>	The FlexIO SPI eDMA type/handle table out of range.

### 15.7.8.4.2 status\_t FLEXIO\_UART\_TransferSendEDMA ( [FLEXIO\\_UART\\_Type](#) \* *base*, [flexio\\_uart\\_edma\\_handle\\_t](#) \* *handle*, [flexio\\_uart\\_transfer\\_t](#) \* *xfer* )

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data have been sent out, the send callback function is called.

Parameters

<i>base</i>	pointer to <a href="#">FLEXIO_UART_Type</a>
<i>handle</i>	UART handle pointer.
<i>xfer</i>	UART eDMA transfer structure, see <a href="#">flexio_uart_transfer_t</a> .

Return values

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

### 15.7.8.4.3 status\_t FLEXIO\_UART\_TransferReceiveEDMA ( [FLEXIO\\_UART\\_Type](#) \* *base*, [flexio\\_uart\\_edma\\_handle\\_t](#) \* *handle*, [flexio\\_uart\\_transfer\\_t](#) \* *xfer* )

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data have been received, the receive callback function is called.

Parameters

<i>base</i>	pointer to <a href="#">FLEXIO_UART_Type</a>
<i>handle</i>	Pointer to flexio_uart_edma_handle_t structure
<i>xfer</i>	UART eDMA transfer structure, see <a href="#">flexio_uart_transfer_t</a> .

Return values

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

#### **15.7.8.4.4 void FLEXIO\_UART\_TransferAbortSendEDMA ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_edma\_handle\_t \* *handle* )**

This function aborts sent data which using eDMA.

Parameters

<i>base</i>	pointer to <a href="#">FLEXIO_UART_Type</a>
<i>handle</i>	Pointer to flexio_uart_edma_handle_t structure

#### **15.7.8.4.5 void FLEXIO\_UART\_TransferAbortReceiveEDMA ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_edma\_handle\_t \* *handle* )**

This function aborts the receive data which using eDMA.

Parameters

<i>base</i>	pointer to <a href="#">FLEXIO_UART_Type</a>
<i>handle</i>	Pointer to flexio_uart_edma_handle_t structure

#### **15.7.8.4.6 status\_t FLEXIO\_UART\_TransferGetSendCountEDMA ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

This function gets the number of bytes still not sent out.

## FlexIO UART Driver

Parameters

<i>base</i>	pointer to <a href="#">FLEXIO_UART_Type</a>
<i>handle</i>	Pointer to flexio_uart_edma_handle_t structure
<i>count</i>	Number of bytes sent so far by the non-blocking transaction.

**15.7.8.4.7 status\_t FLEXIO\_UART\_TransferGetReceiveCountEDMA ( [FLEXIO\\_UART\\_Type](#) \* *base*, [flexio\\_uart\\_edma\\_handle\\_t](#) \* *handle*, [size\\_t](#) \* *count* )**

This function gets the number of bytes still not received.

Parameters

<i>base</i>	pointer to <a href="#">FLEXIO_UART_Type</a>
<i>handle</i>	Pointer to flexio_uart_edma_handle_t structure
<i>count</i>	Number of bytes sent so far by the non-blocking transaction.

## 15.7.9 FlexIO DMA UART Driver

### 15.7.9.1 Overview

#### Data Structures

- struct `flexio_uart_dma_handle_t`  
*UART DMA handle.* [More...](#)

#### TypeDefs

- typedef void(\* `flexio_uart_dma_transfer_callback_t` )(`FLEXIO_UART_Type` \*base, `flexio_uart_dma_handle_t` \*handle, `status_t` status, `void` \*userData)  
*UART transfer callback function.*

#### eDMA transactional

- `status_t FLEXIO_UART_TransferCreateHandleDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_dma_handle_t` \*handle, `flexio_uart_dma_transfer_callback_t` callback, `void` \*userData, `dma_handle_t` \*txDmaHandle, `dma_handle_t` \*rxDmaHandle)  
*Initializes the FLEXIO\_UART handle which is used in transactional functions.*
- `status_t FLEXIO_UART_TransferSendDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_dma_handle_t` \*handle, `flexio_uart_transfer_t` \*xfer)  
*Sends data using DMA.*
- `status_t FLEXIO_UART_TransferReceiveDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_dma_handle_t` \*handle, `flexio_uart_transfer_t` \*xfer)  
*Receives data using DMA.*
- `void FLEXIO_UART_TransferAbortSendDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_dma_handle_t` \*handle)  
*Aborts the sent data which using DMA.*
- `void FLEXIO_UART_TransferAbortReceiveDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_dma_handle_t` \*handle)  
*Aborts the receive data which using DMA.*
- `status_t FLEXIO_UART_TransferGetSendCountDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_dma_handle_t` \*handle, `size_t` \*count)  
*Gets the number of bytes still not sent out.*
- `status_t FLEXIO_UART_TransferGetReceiveCountDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_dma_handle_t` \*handle, `size_t` \*count)  
*Gets the number of bytes still not received.*

## FlexIO UART Driver

### 15.7.9.2 Data Structure Documentation

#### 15.7.9.2.1 struct \_flexio\_uart\_dma\_handle

##### Data Fields

- **flexio\_uart\_dma\_transfer\_callback\_t callback**  
*Callback function.*
- **void \*userData**  
*UART callback function parameter.*
- **size\_t txSize**  
*Total bytes to be sent.*
- **size\_t rxSize**  
*Total bytes to be received.*
- **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.*

#### 15.7.9.2.1.1 Field Documentation

15.7.9.2.1.1.1 **flexio\_uart\_dma\_transfer\_callback\_t flexio\_uart\_dma\_handle\_t::callback**

15.7.9.2.1.1.2 **void\* flexio\_uart\_dma\_handle\_t::userData**

15.7.9.2.1.1.3 **size\_t flexio\_uart\_dma\_handle\_t::txSize**

15.7.9.2.1.1.4 **size\_t flexio\_uart\_dma\_handle\_t::rxSize**

15.7.9.2.1.1.5 **dma\_handle\_t\* flexio\_uart\_dma\_handle\_t::txDmaHandle**

15.7.9.2.1.1.6 **dma\_handle\_t\* flexio\_uart\_dma\_handle\_t::rxDmaHandle**

15.7.9.2.1.1.7 **volatile uint8\_t flexio\_uart\_dma\_handle\_t::txState**

#### 15.7.9.3 Typedef Documentation

15.7.9.3.1 **typedef void(\* flexio\_uart\_dma\_transfer\_callback\_t)(FLEXIO\_UART\_Type \*base, flexio\_uart\_dma\_handle\_t \*handle, status\_t status, void \*userData)**

#### 15.7.9.4 Function Documentation

15.7.9.4.1 **status\_t FLEXIO\_UART\_TransferCreateHandleDMA ( FLEXIO\_UART\_Type \* base, flexio\_uart\_dma\_handle\_t \* handle, flexio\_uart\_dma\_transfer\_callback\_t callback, void \* userData, dma\_handle\_t \* txDmaHandle, dma\_handle\_t \* rxDmaHandle )**

Parameters

<i>base</i>	Pointer to <a href="#">FLEXIO_UART_Type</a> structure.
<i>handle</i>	Pointer to <a href="#">flexio_uart_dma_handle_t</a> structure.
<i>callback</i>	FlexIO UART callback, NULL means no callback.
<i>userData</i>	User callback function data.
<i>txDmaHandle</i>	User requested DMA handle for TX DMA transfer.
<i>rxDmaHandle</i>	User requested DMA handle for RX DMA transfer.

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_OutOfRange</i>	The FlexIO UART DMA type/handle table out of range.

#### 15.7.9.4.2 status\_t FLEXIO\_UART\_TransferSendDMA ( [FLEXIO\\_UART\\_Type](#) \* *base*, [flexio\\_uart\\_dma\\_handle\\_t](#) \* *handle*, [flexio\\_uart\\_transfer\\_t](#) \* *xfer* )

This function send data using DMA, this is non-blocking function, which return right away. When all data have been sent out, the send callback function is called.

Parameters

<i>base</i>	Pointer to <a href="#">FLEXIO_UART_Type</a> structure
<i>handle</i>	Pointer to <a href="#">flexio_uart_dma_handle_t</a> structure
<i>xfer</i>	FLEXIO_UART DMA transfer structure, see <a href="#">flexio_uart_transfer_t</a> .

Return values

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

#### 15.7.9.4.3 status\_t FLEXIO\_UART\_TransferReceiveDMA ( [FLEXIO\\_UART\\_Type](#) \* *base*, [flexio\\_uart\\_dma\\_handle\\_t](#) \* *handle*, [flexio\\_uart\\_transfer\\_t](#) \* *xfer* )

This function receives data using DMA. This is non-blocking function, which returns right away. When all data have been received, the receive callback function is called.

## FlexIO UART Driver

Parameters

<i>base</i>	Pointer to <a href="#">FLEXIO_UART_Type</a> structure
<i>handle</i>	Pointer to <a href="#">flexio_uart_dma_handle_t</a> structure
<i>xfer</i>	FLEXIO_UART DMA transfer structure, see <a href="#">flexio_uart_transfer_t</a> .

Return values

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

**15.7.9.4.4 void FLEXIO\_UART\_TransferAbortSendDMA ( [FLEXIO\\_UART\\_Type](#) \* *base*, [flexio\\_uart\\_dma\\_handle\\_t](#) \* *handle* )**

This function aborts the sent data which using DMA.

Parameters

<i>base</i>	Pointer to <a href="#">FLEXIO_UART_Type</a> structure
<i>handle</i>	Pointer to <a href="#">flexio_uart_dma_handle_t</a> structure

**15.7.9.4.5 void FLEXIO\_UART\_TransferAbortReceiveDMA ( [FLEXIO\\_UART\\_Type](#) \* *base*, [flexio\\_uart\\_dma\\_handle\\_t](#) \* *handle* )**

This function aborts the receive data which using DMA.

Parameters

<i>base</i>	Pointer to <a href="#">FLEXIO_UART_Type</a> structure
<i>handle</i>	Pointer to <a href="#">flexio_uart_dma_handle_t</a> structure

**15.7.9.4.6 status\_t FLEXIO\_UART\_TransferGetSendCountDMA ( [FLEXIO\\_UART\\_Type](#) \* *base*, [flexio\\_uart\\_dma\\_handle\\_t](#) \* *handle*, [size\\_t](#) \* *count* )**

This function gets the number of bytes still not sent out.

Parameters

<i>base</i>	Pointer to <a href="#">FLEXIO_UART_Type</a> structure
<i>handle</i>	Pointer to <code>flexio_uart_dma_handle_t</code> structure
<i>count</i>	Number of bytes sent so far by the non-blocking transaction.

#### 15.7.9.4.7 `status_t FLEXIO_UART_TransferGetReceiveCountDMA ( FLEXIO_UART_Type * base, flexio_uart_dma_handle_t * handle, size_t * count )`

This function gets the number of bytes still not received.

Parameters

<i>base</i>	Pointer to <a href="#">FLEXIO_UART_Type</a> structure
<i>handle</i>	Pointer to <code>flexio_uart_dma_handle_t</code> structure
<i>count</i>	Number of bytes received so far by the non-blocking transaction.



# Chapter 16

## GPIO: General-Purpose Input/Output Driver

### 16.1 Overview

#### Modules

- FGPIO Driver
- GPIO Driver

#### Data Structures

- struct `gpio_pin_config_t`  
*The GPIO pin configuration structure. [More...](#)*

#### Enumerations

- enum `gpio_pin_direction_t` {  
  `kGPIO_DigitalInput` = 0U,  
  `kGPIO_DigitalOutput` = 1U }  
*GPIO direction definition.*

#### Driver version

- #define `FSL_GPIO_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 0))  
*GPIO driver version 2.1.0.*

### 16.2 Data Structure Documentation

#### 16.2.1 struct `gpio_pin_config_t`

Every pin can only be configured as either output pin or input pin at a time. If configured as a input pin, then leave the `outputConfig` unused Note : 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 default output logic, no use in input.*

## Enumeration Type Documentation

### 16.3 Macro Definition Documentation

16.3.1 `#define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

### 16.4 Enumeration Type Documentation

#### 16.4.1 `enum gpio_pin_direction_t`

Enumerator

*kGPIO\_DigitalInput* Set current pin as digital input.

*kGPIO\_DigitalOutput* Set current pin as digital output.

## 16.5 GPIO Driver

### 16.5.1 Overview

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

### 16.5.2 Typical use case

#### 16.5.2.1 Output Operation

```
/* Output pin configuration */
gpio_pin_config_t led_config =
{
    kGpioDigitalOutput,
    1,
};

/* Sets the configuration */
GPIO_PinInit(GPIO_LED, LED_PINNUM, &led_config);
```

#### 16.5.2.2 Input Operation

```
/* Input pin configuration */
PORT_SetPinInterruptConfig(BOARD_SW2_PORT, BOARD_SW2_GPIO_PIN,
                           kPORT_InterruptFallingEdge);
NVIC_EnableIRQ(BOARD_SW2_IRQ);
gpio_pin_config_t sw1_config =
{
    kGpioDigitalInput,
    0,
};

/* Sets the input pin configuration */
GPIO_PinInit(GPIO_SW1, SW1_PINNUM, &sw1_config);
```

## 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\_WritePinOutput** (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\_SetPinsOutput** (GPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 1.*
- static void **GPIO\_ClearPinsOutput** (GPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 0.*
- static void **GPIO\_TogglePinsOutput** (GPIO\_Type \*base, uint32\_t mask)  
*Reverses current output logic of the multiple GPIO pins.*

### GPIO Input Operations

- static uint32\_t [GPIO\\_ReadPinInput](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current input value of the whole GPIO port.*

### GPIO Interrupt

- uint32\_t [GPIO\\_GetPinsInterruptFlags](#) (GPIO\_Type \*base)  
*Reads whole GPIO port interrupt status flag.*
- void [GPIO\\_ClearPinsInterruptFlags](#) (GPIO\_Type \*base, uint32\_t mask)  
*Clears multiple GPIO pin interrupt status flag.*

### 16.5.3 Function Documentation

#### 16.5.3.1 void [GPIO\\_PinInit](#) ( **GPIO\_Type** \* *base*, **uint32\_t** *pin*, **const gpio\_pin\_config\_t** \* *config* )

To initialize the GPIO, define a pin configuration, either input or output, in the user file. Then, call the [GPIO\\_PinInit\(\)](#) function.

This is an example to define an input pin or 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

#### 16.5.3.2 static void [GPIO\\_WritePinOutput](#) ( **GPIO\_Type** \* *base*, **uint32\_t** *pin*, **uint8\_t** *output* ) [inline], [static]

Parameters

<i>base</i>	GPIO peripheral base pointer(GPIOA, GPIOB, GPIOC, and so on.)
<i>pin</i>	GPIO pin number
<i>output</i>	GPIO pin output logic level. <ul style="list-style-type: none"> <li>• 0: corresponding pin output low-logic level.</li> <li>• 1: corresponding pin output high-logic level.</li> </ul>

**16.5.3.3 static void GPIO\_SetPinsOutput ( **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

**16.5.3.4 static void GPIO\_ClearPinsOutput ( **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

**16.5.3.5 static void GPIO\_TogglePinsOutput ( **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

**16.5.3.6 static **uint32\_t** GPIO\_ReadPinInput ( **GPIO\_Type** \* *base*, **uint32\_t** *pin* )  
[inline], [static]**

## GPIO Driver

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>
-------------	---

### 16.5.3.7 `uint32_t GPIO_GetPinsInterruptFlags ( 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>Current</i>	GPIO port interrupt status flag, for example, 0x00010001 means the pin 0 and 17 have the interrupt.
----------------	---

### 16.5.3.8 `void GPIO_ClearPinsInterruptFlags ( 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

## 16.6 FGPIO Driver

This chapter 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.

### 16.6.1 Typical use case

#### 16.6.1.1 Output Operation

```
/* Output pin configuration */
gpio_pin_config_t led_config =
{
    kGpioDigitalOutput,
    1,
};

/* Sets the configuration */
GPIO_PinInit(GPIO_LED, LED_PINNUM, &led_config);
```

#### 16.6.1.2 Input Operation

```
/* Input pin configuration */
PORT_SetPinInterruptConfig(BOARD_SW2_PORT, BOARD_SW2_FGPIO_PIN,
                           kPORT_InterruptFallingEdge);
NVIC_EnableIRQ(BOARD_SW2_IRQ);
gpio_pin_config_t sw1_config =
{
    kGpioDigitalInput,
    0,
};

/* Sets the input pin configuration */
GPIO_PinInit(GPIO_SW1, SW1_PINNUM, &sw1_config);
```



# **Chapter 17**

## **I2C: Inter-Integrated Circuit Driver**

### **17.1 Overview**

#### **Modules**

- I2C DMA Driver
- I2C Driver
- I2C FreeRTOS Driver
- I2C eDMA Driver
- I2C µCOS/II Driver
- I2C µCOS/III Driver

### 17.2 I2C Driver

#### 17.2.1 Overview

The KSDK provides a peripheral driver for the Inter-Integrated Circuit (I2C) module of Kinetis devices.

The I2C driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target 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 the knowledge of 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 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 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.

#### 17.2.2 Typical use case

##### 17.2.2.1 Master Operation in functional method

```
i2c_master_config_t masterConfig;
uint8_t status;
status_t result = kStatus_Success;
uint8_t txBuff[BUFFER_SIZE];

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

/* Send start and slave address. */
I2C_MasterStart(EXAMPLE_I2C_MASTER_BASEADDR, 7-bit slave address,
    kI2C_Write/kI2C_Read);

/* Wait address sent out. */
while(!((status = I2C_GetStatusFlag(EXAMPLE_I2C_MASTER_BASEADDR)) & kI2C_IntPendingFlag))
{
}

if(status & kI2C_ReceiveNakFlag)
{
    return kStatus_I2C_Nak;
}

result = I2C_MasterWriteBlocking(EXAMPLE_I2C_MASTER_BASEADDR, txBuff, BUFFER_SIZE);

if(result)
{
    /* If error occurs, send STOP. */
```

```

    I2C_MasterStop(EXAMPLE_I2C_MASTER_BASEADDR, kI2CStop);
    return result;
}

while (!(I2C_GetStatusFlag(EXAMPLE_I2C_MASTER_BASEADDR) & kI2C_IntPendingFlag))
{
}

/* Wait all data sent out, send STOP. */
I2C_MasterStop(EXAMPLE_I2C_MASTER_BASEADDR, kI2CStop);

```

### 17.2.2.2 Master Operation in interrupt transactional method

```

i2c_master_handle_t g_m_handle;
volatile bool g_MasterCompletionFlag = false;
i2c_master_config_t masterConfig;
uint8_t status;
status_t result = kStatus_Success;
uint8_t txBuff[BUFFER_SIZE];
i2c_master_transfer_t masterXfer;

static void i2c_master_callback(I2C_Type *base, i2c_master_handle_t *handle, status_t status, void *userData)
{
    /* Signal transfer success when received success status. */
    if (status == kStatus_Success)
    {
        g_MasterCompletionFlag = true;
    }
}

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

masterXfer.slaveAddress = I2C_MASTER_SLAVE_ADDR_7BIT;
masterXfer.direction = kI2C_Write;
masterXfer.subaddress = NULL;
masterXfer.subaddressSize = 0;
masterXfer.data = txBuff;
masterXfer.dataSize = BUFFER_SIZE;
masterXfer.flags = kI2C_TransferDefaultFlag;

I2C_MasterTransferCreateHandle(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_handle,
    i2c_master_callback, NULL);
I2C_MasterTransferNonBlocking(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_handle, &masterXfer);

/* Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;

```

### 17.2.2.3 Master Operation in DMA transactional method

```

i2c_master_dma_handle_t g_m_dma_handle;
dma_handle_t dmaHandle;
volatile bool g_MasterCompletionFlag = false;
i2c_master_config_t masterConfig;

```

## I2C Driver

```
uint8_t txBuff[BUFFER_SIZE];
i2c_master_transfer_t masterXfer;

static void i2c_master_callback(I2C_Type *base, i2c_master_dma_handle_t *handle, status_t status, void *userData)
{
    /* Signal transfer success when received success status. */
    if (status == kStatus_Success)
    {
        g_MasterCompletionFlag = true;
    }
}

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

masterXfer.slaveAddress = I2C_MASTER_SLAVE_ADDR_7BIT;
masterXfer.direction = kI2C_Write;
masterXfer.subaddress = NULL;
masterXfer.subaddressSize = 0;
masterXfer.data = txBuff;
masterXfer.dataSize = BUFFER_SIZE;
masterXfer.flags = kI2C_TransferDefaultFlag;

DMAMGR_RequestChannel((dma_request_source_t)DMA_REQUEST_SRC, 0, &dmaHandle);

I2C_MasterTransferCreateHandleDMA(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_dma_handle, i2c_master_callback, NULL, &dmaHandle);
I2C_MasterTransferDMA(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_dma_handle, &masterXfer);

/* Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

### 17.2.2.4 Slave Operation in functional method

```
i2c_slave_config_t slaveConfig;
uint8_t status;
status_t result = kStatus_Success;

I2C_SlaveGetDefaultConfig(&slaveConfig); /*default configuration 7-bit addressing mode*/
slaveConfig.slaveAddr = 7-bit address
slaveConfig.addressingMode = kI2C_Address7bit/
    kI2C_RangeMatch;
I2C_SlaveInit(EXAMPLE_I2C_SLAVE_BASEADDR, &slaveConfig);

/* Wait address match. */
while (!(status = I2C_GetStatusFlag(EXAMPLE_I2C_SLAVE_BASEADDR)) & kI2C_AddressMatchFlag)
{

}

/* Slave transmit, master reading from slave. */
if (status & kI2C_TransferDirectionFlag)
{
    result = I2C_SlaveWriteBlocking(EXAMPLE_I2C_SLAVE_BASEADDR);
}
else
{
```

```

    I2C_SlaveReadBlocking(EXAMPLE_I2C_SLAVE_BASEADDR);
}

return result;
}

17.2.2.5 Slave Operation in interrupt transactional method

i2c_slave_config_t slaveConfig;
i2c_slave_handle_t g_s_handle;
volatile bool g_SlaveCompletionFlag = false;

static void i2c_slave_callback(I2C_Type *base, i2c_slave_transfer_t *xfer, void *
    userData)
{
    switch (xfer->event)
    {
        /* Transmit request */
        case kI2C_SlaveTransmitEvent:
            /* Update information for transmit process */
            xfer->data = g_slave_buff;
            xfer->dataSize = I2C_DATA_LENGTH;
            break;

        /* Receive request */
        case kI2C_SlaveReceiveEvent:
            /* Update information for received process */
            xfer->data = g_slave_buff;
            xfer->dataSize = I2C_DATA_LENGTH;
            break;

        /* Transfer done */
        case kI2C_SlaveCompletionEvent:
            g_SlaveCompletionFlag = true;
            break;

        default:
            g_SlaveCompletionFlag = true;
            break;
    }
}

I2C_SlaveGetDefaultConfig(&slaveConfig); /*default configuration 7-bit addressing
    mode*/
slaveConfig.slaveAddr = 7-bit address
slaveConfig.addressingMode = kI2C_Address7bit/
    kI2C_RangeMatch;

I2C_SlaveInit(EXAMPLE_I2C_SLAVE_BASEADDR, &slaveConfig);

I2C_SlaveTransferCreateHandle(EXAMPLE_I2C_SLAVE_BASEADDR, &g_s_handle,
    i2c_slave_callback, NULL);

I2C_SlaveTransferNonBlocking(EXAMPLE_I2C_SLAVE_BASEADDR, &g_s_handle,
    kI2C_SlaveCompletionEvent);

/* Wait for transfer completed. */
while (!g_SlaveCompletionFlag)
{
}
g_SlaveCompletionFlag = false;

```

## I2C Driver

### 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...](#)

### 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 `_i2c_status` {  
  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) }  
*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 }  
*I2C peripheral flags.*
- enum `_i2c_interrupt_enable` { kI2C\_GlobalInterruptEnable = I2C\_C1\_IICIE\_MASK }  
*I2C feature interrupt source.*

- enum `i2c_direction_t` {
   
  `kI2C_Write` = 0x0U,
   
  `kI2C_Read` = 0x1U }
   
*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_SlaveCompletionEvent` = 0x20U,
   
  `kI2C_SlaveAllEvents` }
   
*Set of events sent to the callback for nonblocking slave transfers.*

## Driver version

- #define `FSL_I2C_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 1)`)
   
*I2C driver version 2.0.1.*

## 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)
   
*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.*
- 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.*

## I2C Driver

### 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 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)`  
*Performs a polling send transaction on the I2C bus without a STOP signal.*
- `status_t I2C_MasterReadBlocking (I2C_Type *base, uint8_t *rxBuff, size_t rxSize)`  
*Performs a polling receive transaction on the I2C bus with a STOP signal.*
- `status_t I2C_SlaveWriteBlocking (I2C_Type *base, const uint8_t *txBuff, size_t txSize)`  
*Performs a polling send transaction on the I2C bus.*
- `void 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.*
- void [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.*

### 17.2.3 Data Structure Documentation

#### 17.2.3.1 struct i2c\_master\_config\_t

##### Data Fields

- bool [enableMaster](#)

*Enables the I2C peripheral at initialization time.*
- uint32\_t [baudRate\\_Bps](#)

*Baud rate configuration of I2C peripheral.*
- uint8\_t [glitchFilterWidth](#)

*Controls the width of the glitch.*

## I2C Driver

### 17.2.3.1.0.1 Field Documentation

17.2.3.1.0.1.1 `bool i2c_master_config_t::enableMaster`

17.2.3.1.0.1.2 `uint32_t i2c_master_config_t::baudRate_Bps`

17.2.3.1.0.1.3 `uint8_t i2c_master_config_t::glitchFilterWidth`

### 17.2.3.2 `struct i2c_slave_config_t`

#### Data Fields

- `bool enableSlave`  
*Enables the I2C peripheral at initialization time.*
- `bool enableGeneralCall`  
*Enable 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`  
*Slave address configuration.*
- `uint16_t upperAddress`  
*Maximum boundary slave address used in range matching mode.*
- `i2c_slave_address_mode_t addressingMode`  
*Addressing mode configuration of i2c\_slave\_address\_mode\_config\_t.*

### 17.2.3.2.0.2 Field Documentation

17.2.3.2.0.2.1 `bool i2c_slave_config_t::enableSlave`

17.2.3.2.0.2.2 `bool i2c_slave_config_t::enableGeneralCall`

17.2.3.2.0.2.3 `bool i2c_slave_config_t::enableWakeUp`

17.2.3.2.0.2.4 `bool i2c_slave_config_t::enableBaudRateCtl`

17.2.3.2.0.2.5 `uint16_t i2c_slave_config_t::slaveAddress`

17.2.3.2.0.2.6 `uint16_t i2c_slave_config_t::upperAddress`

17.2.3.2.0.2.7 `i2c_slave_address_mode_t i2c_slave_config_t::addressingMode`

### 17.2.3.3 `struct i2c_master_transfer_t`

#### Data Fields

- `uint32_t flags`  
*Transfer flag which controls the transfer.*
- `uint8_t slaveAddress`  
*7-bit slave address.*

- **i2c\_direction\_t direction**  
*Transfer direction, read or write.*
- **uint32\_t subaddress**  
*Sub address.*
- **uint8\_t subaddressSize**  
*Size of command buffer.*
- **uint8\_t \*volatile data**  
*Transfer buffer.*
- **volatile size\_t dataSize**  
*Transfer size.*

### 17.2.3.3.0.3 Field Documentation

#### 17.2.3.3.0.3.1 uint32\_t i2c\_master\_transfer\_t::flags

#### 17.2.3.3.0.3.2 uint8\_t i2c\_master\_transfer\_t::slaveAddress

#### 17.2.3.3.0.3.3 i2c\_direction\_t i2c\_master\_transfer\_t::direction

#### 17.2.3.3.0.3.4 uint32\_t i2c\_master\_transfer\_t::subaddress

Transferred MSB first.

#### 17.2.3.3.0.3.5 uint8\_t i2c\_master\_transfer\_t::subaddressSize

#### 17.2.3.3.0.3.6 uint8\_t\* volatile i2c\_master\_transfer\_t::data

#### 17.2.3.3.0.3.7 volatile size\_t i2c\_master\_transfer\_t::dataSize

### 17.2.3.4 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**  
*Transfer state maintained during transfer.*
- **i2c\_master\_transfer\_callback\_t completionCallback**  
*Callback function called when transfer finished.*
- **void \*userData**  
*Callback parameter passed to callback function.*

## I2C Driver

### 17.2.3.4.0.4 Field Documentation

17.2.3.4.0.4.1 `i2c_master_transfer_t i2c_master_handle_t::transfer`

17.2.3.4.0.4.2 `size_t i2c_master_handle_t::transferSize`

17.2.3.4.0.4.3 `uint8_t i2c_master_handle_t::state`

17.2.3.4.0.4.4 `i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback`

17.2.3.4.0.4.5 `void* i2c_master_handle_t::userData`

### 17.2.3.5 struct `i2c_slave_transfer_t`

#### Data Fields

- `i2c_slave_transfer_event_t event`  
*Reason the callback is being invoked.*
- `uint8_t *volatile data`  
*Transfer buffer.*
- `volatile size_t dataSize`  
*Transfer size.*
- `status_t completionStatus`  
*Success or error code describing how the transfer completed.*
- `size_t transferredCount`  
*Number of bytes actually transferred since start or last repeated start.*

### 17.2.3.5.0.5 Field Documentation

17.2.3.5.0.5.1 `i2c_slave_transfer_event_t i2c_slave_transfer_t::event`

17.2.3.5.0.5.2 `uint8_t* volatile i2c_slave_transfer_t::data`

17.2.3.5.0.5.3 `volatile size_t i2c_slave_transfer_t::dataSize`

17.2.3.5.0.5.4 `status_t i2c_slave_transfer_t::completionStatus`

Only applies for `kI2C_SlaveCompletionEvent`.

17.2.3.5.0.5.5 `size_t i2c_slave_transfer_t::transferredCount`

### 17.2.3.6 struct `_i2c_slave_handle`

I2C slave handle typedef.

#### Data Fields

- `bool isBusy`  
*Whether transfer is busy.*
- `i2c_slave_transfer_t transfer`

- *I2C slave transfer copy.*
- `uint32_t eventMask`  
*Mask of enabled events.*
- `i2c_slave_transfer_callback_t callback`  
*Callback function called at transfer event.*
- `void *userData`  
*Callback parameter passed to callback.*

### 17.2.3.6.0.6 Field Documentation

17.2.3.6.0.6.1 `bool i2c_slave_handle_t::isBusy`

17.2.3.6.0.6.2 `i2c_slave_transfer_t i2c_slave_handle_t::transfer`

17.2.3.6.0.6.3 `uint32_t i2c_slave_handle_t::eventMask`

17.2.3.6.0.6.4 `i2c_slave_transfer_callback_t i2c_slave_handle_t::callback`

17.2.3.6.0.6.5 `void* i2c_slave_handle_t::userData`

### 17.2.4 Macro Definition Documentation

17.2.4.1 `#define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))`

### 17.2.5 Typedef Documentation

17.2.5.1 `typedef void(* i2c_master_transfer_callback_t)(I2C_Type *base, i2c_master_handle_t *handle, status_t status, void *userData)`

17.2.5.2 `typedef void(* i2c_slave_transfer_callback_t)(I2C_Type *base, i2c_slave_transfer_t *xfer, void *userData)`

### 17.2.6 Enumeration Type Documentation

#### 17.2.6.1 `enum _i2c_status`

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` Wait event timeout.

## I2C Driver

### 17.2.6.2 enum \_i2c\_flags

The following status register flags can be cleared:

- `kI2C_ArbitrationLostFlag`
- `kI2C_IntPendingFlag`
- `#kI2C_StartDetectFlag`
- `#kI2C_StopDetectFlag`

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.  
`kI2C_TransferDirectionFlag` I2C transfer direction flag.  
`kI2C_RangeAddressMatchFlag` I2C range address match flag.  
`kI2C_ArbitrationLostFlag` I2C arbitration lost flag.  
`kI2C_BusBusyFlag` I2C bus busy flag.  
`kI2C_AddressMatchFlag` I2C address match flag.  
`kI2C_TransferCompleteFlag` I2C transfer complete flag.

### 17.2.6.3 enum \_i2c\_interrupt\_enable

Enumerator

- `kI2C_GlobalInterruptEnable` I2C global interrupt.

### 17.2.6.4 enum i2c\_direction\_t

Enumerator

- `kI2C_Write` Master transmit to slave.  
`kI2C_Read` Master receive from slave.

### 17.2.6.5 enum i2c\_slave\_address\_mode\_t

Enumerator

- `kI2C_Address7bit` 7-bit addressing mode.  
`kI2C_RangeMatch` Range address match addressing mode.

### 17.2.6.6 enum \_i2c\_master\_transfer\_flags

Enumerator

**kI2C\_TransferDefaultFlag** Transfer starts with a start signal, stops with a stop signal.

**kI2C\_TransferNoStartFlag** Transfer starts without a start signal.

**kI2C\_TransferRepeatedStartFlag** Transfer starts with a repeated start signal.

**kI2C\_TransferNoStopFlag** Transfer ends without a stop signal.

### 17.2.6.7 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\(\)](#) in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note

These enumerations are meant to be OR'd together to form a bit mask of events.

Enumerator

**kI2C\_SlaveAddressMatchEvent** Received the slave address after a start or repeated start.

**kI2C\_SlaveTransmitEvent** Callback is requested to provide data to transmit (slave-transmitter role).

**kI2C\_SlaveReceiveEvent** Callback is requested to provide a buffer in which to place received data (slave-receiver role).

**kI2C\_SlaveTransmitAckEvent** Callback needs to either transmit an ACK or NACK.

**kI2C\_SlaveCompletionEvent** A stop was detected or finished transfer, completing the transfer.

**kI2C\_SlaveAllEvents** Bit mask of all available events.

## 17.2.7 Function Documentation

### 17.2.7.1 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 to use the I2C driver, or any operation to the I2C module may cause a hard fault because clock is not enabled. The configuration structure can be filled by user from scratch, or be set with default values by [I2C\\_MasterGetDefaultConfig\(\)](#). After calling this API, the master is ready to transfer. Example:

## I2C Driver

```
* 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>	pointer to master configuration structure
<i>srcClock_Hz</i>	I2C peripheral clock frequency in Hz

### 17.2.7.2 void I2C\_SlaveInit ( I2C\_Type \* *base*, const i2c\_slave\_config\_t \* *slaveConfig* )

Call this API to ungate the I2C clock and initializes the I2C with slave configuration.

#### Note

This API should be called at the beginning of the application to use the I2C driver, or 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 can be filled by the user.  
Example

```
* i2c_slave_config_t config = {  
* .enableSlave = true,  
* .enableGeneralCall = false,  
* .addressingMode = kI2C_Address7bit,  
* .slaveAddress = 0x1DU,  
* .enableWakeUp = false,  
* .enablehighDrive = false,  
* .enableBaudRateCtl = false  
* };  
* I2C_SlaveInit(I2C0, &config);  
*
```

### Parameters

<i>base</i>	I2C base pointer
<i>slaveConfig</i>	pointer to slave configuration structure

### 17.2.7.3 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
-------------	------------------

#### 17.2.7.4 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
-------------	------------------

#### 17.2.7.5 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 I2C\_MasterConfigure(), or modify some fields of the structure before calling I2C\_MasterConfigure(). Example:

```
* i2c_master_config_t config;
* I2C_MasterGetDefaultConfig(&config);
*
```

Parameters

<i>masterConfig</i>	Pointer to the master configuration structure.
---------------------	--

#### 17.2.7.6 void I2C\_SlaveGetDefaultConfig ( i2c\_slave\_config\_t \* *slaveConfig* )

The purpose of this API is to get the configuration structure initialized for use in I2C\_SlaveConfigure(). Modify fields of the structure before calling the I2C\_SlaveConfigure(). Example:

```
* i2c_slave_config_t config;
* I2C_SlaveGetDefaultConfig(&config);
*
```

Parameters

## I2C Driver

<i>slaveConfig</i>	Pointer to the slave configuration structure.
--------------------	---

### 17.2.7.7 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 module, false to disable module

### 17.2.7.8 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.

### 17.2.7.9 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.

### 17.2.7.10 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>

#### 17.2.7.11 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>

#### 17.2.7.12 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>

#### 17.2.7.13 void I2C\_DisableInterrupts ( I2C\_Type \* *base*, uint32\_t *mask* )

## I2C Driver

Parameters

<i>base</i>	I2C base pointer
<i>mask</i>	interrupt source The parameter can be combination of the following source if defined: <ul style="list-style-type: none"><li>• kI2C_GlobalInterruptEnable</li><li>• kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable</li><li>• kI2C_SdaTimeoutInterruptEnable</li></ul>

### 17.2.7.14 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

### 17.2.7.15 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

### 17.2.7.16 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.

#### 17.2.7.17 **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.

#### 17.2.7.18 **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.

#### 17.2.7.19 **status\_t I2C\_MasterWriteBlocking ( I2C\_Type \* *base*, const uint8\_t \* *txBuff*, size\_t *txSize* )**

## I2C Driver

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_Arbitration-Lost</i>	Transfer error, arbitration lost.
<i>kStatus_I2C_Nak</i>	Transfer error, receive NAK during transfer.

### 17.2.7.20 **status\_t I2C\_MasterReadBlocking ( I2C\_Type \* *base*, uint8\_t \* *rxBuff*, size\_t *rxSize* )**

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.

Return values

<i>kStatus_Success</i>	Successfully complete the data transmission.
<i>kStatus_I2C_Timeout</i>	Send stop signal failed, timeout.

### 17.2.7.21 **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.

**17.2.7.22 void 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.

**17.2.7.23 status\_t I2C\_MasterTransferBlocking ( I2C\_Type \* *base*, i2c\_master\_transfer\_t \* *xfer* )**

## I2C Driver

### 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.

### 17.2.7.24 void I2C\_MasterTransferCreateHandle ( *I2C\_Type \* base, i2c\_master\_handle\_t \* handle, i2c\_master\_transfer\_callback\_t callback, void \* userData* )

### Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to <i>i2c_master_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.

### 17.2.7.25 status\_t I2C\_MasterTransferNonBlocking ( *I2C\_Type \* base, i2c\_master\_handle\_t \* handle, i2c\_master\_transfer\_t \* xfer* )

### Note

Calling the API returns immediately after transfer initiates. The user needs to call *I2C\_MasterGetTransferCount* to poll the transfer status to check whether the transfer is finished. If the return status is not *kStatus\_I2C\_Busy*, the transfer is finished.

Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to i2c_master_handle_t structure which stores the transfer state.
<i>xfer</i>	pointer to <a href="#">i2c_master_transfer_t</a> structure.

Return values

<i>kStatus_Success</i>	Successfully start the data transmission.
<i>kStatus_I2C_Busy</i>	Previous transmission still not finished.
<i>kStatus_I2C_Timeout</i>	Transfer error, wait signal timeout.

#### 17.2.7.26 **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.

#### 17.2.7.27 **void 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

**17.2.7.28 void I2C\_MasterTransferHandleIRQ ( I2C\_Type \* *base*, void \* *i2cHandle* )**

Parameters

<i>base</i>	I2C base pointer.
<i>i2cHandle</i>	pointer to i2c_master_handle_t structure.

#### 17.2.7.29 void I2C\_SlaveTransferCreateHandle ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle*, i2c\_slave\_transfer\_callback\_t *callback*, void \* *userData* )

Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to i2c_slave_handle_t structure to store the transfer state.
<i>callback</i>	pointer to user callback function.
<i>userData</i>	user parameter passed to the callback function.

#### 17.2.7.30 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 # <a href="#">i2c_slave_handle_t</a> structure which stores the transfer state.
<i>eventMask</i>	Bit mask formed by OR'ing together <a href="#">i2c_slave_transfer_event_t</a> enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and <a href="#">kI2C_SlaveAllEvents</a> to enable all events.

## I2C Driver

Return values

#kStatus_Success	Slave transfers were successfully started.
kStatus_I2C_Busy	Slave transfers have already been started on this handle.

### 17.2.7.31 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.

### 17.2.7.32 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

kStatus_InvalidArgument	count is Invalid.
kStatus_Success	Successfully return the count.

### 17.2.7.33 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

## 17.3 I2C eDMA Driver

### 17.3.1 Overview

#### Data Structures

- struct [i2c\\_master\\_edma\\_handle\\_t](#)  
*I2C master eDMA transfer structure.* [More...](#)

#### Typedefs

- [typedef void\(\\* i2c\\_master\\_edma\\_transfer\\_callback\\_t \)\(I2C\\_Type \\*base, i2c\\_master\\_edma\\_handle\\_t \\*handle, status\\_t status, void \\*userData\)](#)  
*I2C master eDMA transfer callback typedef.*

#### I2C Block eDMA Transfer Operation

- [void I2C\\_MasterCreateEDMAHandle \(I2C\\_Type \\*base, i2c\\_master\\_edma\\_handle\\_t \\*handle, i2c\\_master\\_edma\\_transfer\\_callback\\_t callback, void \\*userData, edma\\_handle\\_t \\*edmaHandle\)](#)  
*Init the I2C handle which is used in transational functions.*
- [status\\_t I2C\\_MasterTransferEDMA \(I2C\\_Type \\*base, i2c\\_master\\_edma\\_handle\\_t \\*handle, i2c\\_master\\_transfer\\_t \\*xfer\)](#)  
*Performs a master eDMA non-blocking transfer on the I2C bus.*
- [status\\_t I2C\\_MasterTransferGetCountEDMA \(I2C\\_Type \\*base, i2c\\_master\\_edma\\_handle\\_t \\*handle, size\\_t \\*count\)](#)  
*Get master transfer status during a eDMA non-blocking transfer.*
- [void I2C\\_MasterTransferAbortEDMA \(I2C\\_Type \\*base, i2c\\_master\\_edma\\_handle\\_t \\*handle\)](#)  
*Abort a master eDMA non-blocking transfer in a early time.*

### 17.3.2 Data Structure Documentation

#### 17.3.2.1 struct \_i2c\_master\_edma\_handle

I2C master eDMA 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.*
- [edma\\_handle\\_t \\* dmaHandle](#)

## I2C eDMA Driver

The eDMA handler used.

- **i2c\_master\_edma\_transfer\_callback\_t completionCallback**  
Callback function called after eDMA transfer finished.
- **void \*userData**  
Callback parameter passed to callback function.

### 17.3.2.1.0.7 Field Documentation

17.3.2.1.0.7.1 **i2c\_master\_transfer\_t i2c\_master\_edma\_handle\_t::transfer**

17.3.2.1.0.7.2 **size\_t i2c\_master\_edma\_handle\_t::transferSize**

17.3.2.1.0.7.3 **uint8\_t i2c\_master\_edma\_handle\_t::state**

17.3.2.1.0.7.4 **edma\_handle\_t\* i2c\_master\_edma\_handle\_t::dmaHandle**

17.3.2.1.0.7.5 **i2c\_master\_edma\_transfer\_callback\_t i2c\_master\_edma\_handle\_t::completionCallback**

17.3.2.1.0.7.6 **void\* i2c\_master\_edma\_handle\_t::userData**

### 17.3.3 Typedef Documentation

17.3.3.1 **typedef void(\* i2c\_master\_edma\_transfer\_callback\_t)(I2C\_Type \*base, i2c\_master\_edma\_handle\_t \*handle, status\_t status, void \*userData)**

### 17.3.4 Function Documentation

17.3.4.1 **void I2C\_MasterCreateEDMAHandle ( I2C\_Type \* base, i2c\_master\_edma\_handle\_t \* handle, i2c\_master\_edma\_transfer\_callback\_t callback, void \* userData, edma\_handle\_t \* edmaHandle )**

Parameters

<i>base</i>	I2C peripheral base address.
<i>handle</i>	pointer to i2c_master_edma_handle_t structure.
<i>callback</i>	pointer to user callback function.
<i>userData</i>	user param passed to the callback function.
<i>edmaHandle</i>	eDMA handle pointer.

17.3.4.2 **status\_t I2C\_MasterTransferEDMA ( I2C\_Type \* base, i2c\_master\_edma\_handle\_t \* handle, i2c\_master\_transfer\_t \* xfer )**

Parameters

<i>base</i>	I2C peripheral base address.
<i>handle</i>	pointer to i2c_master_edma_handle_t structure.
<i>xfer</i>	pointer to transfer structure of i2c_master_transfer_t.

Return values

<i>kStatus_Success</i>	Sucessully 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.

#### 17.3.4.3 **status\_t I2C\_MasterTransferGetCountEDMA ( I2C\_Type \* *base*, i2c\_master\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

<i>base</i>	I2C peripheral base address.
<i>handle</i>	pointer to i2c_master_edma_handle_t structure.
<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

#### 17.3.4.4 **void I2C\_MasterTransferAbortEDMA ( I2C\_Type \* *base*, i2c\_master\_edma\_handle\_t \* *handle* )**

Parameters

<i>base</i>	I2C peripheral base address.
<i>handle</i>	pointer to i2c_master_edma_handle_t structure.

### 17.4 I2C DMA Driver

#### 17.4.1 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.*

#### 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)  
*Init the I2C handle which is used in transnational 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)  
*Get master transfer status during a dma non-blocking transfer.*
- void [I2C\\_MasterTransferAbortDMA](#) (I2C\_Type \*base, i2c\_master\_dma\_handle\_t \*handle)  
*Abort a master dma non-blocking transfer in a early time.*

#### 17.4.2 Data Structure Documentation

##### 17.4.2.1 struct \_i2c\_master\_dma\_handle

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**  
*Callback function called after dma transfer finished.*
- **void \*userData**  
*Callback parameter passed to callback function.*

#### 17.4.2.1.0.8 Field Documentation

17.4.2.1.0.8.1 **i2c\_master\_transfer\_t i2c\_master\_dma\_handle\_t::transfer**

17.4.2.1.0.8.2 **size\_t i2c\_master\_dma\_handle\_t::transferSize**

17.4.2.1.0.8.3 **uint8\_t i2c\_master\_dma\_handle\_t::state**

17.4.2.1.0.8.4 **dma\_handle\_t\* i2c\_master\_dma\_handle\_t::dmaHandle**

17.4.2.1.0.8.5 **i2c\_master\_dma\_transfer\_callback\_t i2c\_master\_dma\_handle\_t::completionCallback**

17.4.2.1.0.8.6 **void\* i2c\_master\_dma\_handle\_t::userData**

#### 17.4.3 Typedef Documentation

17.4.3.1 **typedef void(\* i2c\_master\_dma\_transfer\_callback\_t)(I2C\_Type \*base, i2c\_master\_dma\_handle\_t \*handle, status\_t status, void \*userData)**

#### 17.4.4 Function Documentation

17.4.4.1 **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 i2c_master_dma_handle_t structure
<i>callback</i>	pointer to user callback function
<i>userData</i>	user param passed to the callback function
<i>dmaHandle</i>	DMA handle pointer

17.4.4.2 **status\_t I2C\_MasterTransferDMA ( I2C\_Type \* *base*, i2c\_master\_dma\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *xfer* )**

## I2C DMA Driver

Parameters

<i>base</i>	I2C peripheral base address
<i>handle</i>	pointer to i2c_master_dma_handle_t structure
<i>xfer</i>	pointer to transfer structure of i2c_master_transfer_t

Return values

<i>kStatus_Success</i>	Sucessully 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.

### 17.4.4.3 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>	pointer to i2c_master_dma_handle_t structure
<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

### 17.4.4.4 void I2C\_MasterTransferAbortDMA ( *I2C\_Type \* base*, *i2c\_master\_dma\_handle\_t \* handle* )

Parameters

<i>base</i>	I2C peripheral base address
<i>handle</i>	pointer to i2c_master_dma_handle_t structure

## 17.5 I2C FreeRTOS Driver

### 17.5.1 Overview

#### Data Structures

- struct [i2c\\_rtos\\_handle\\_t](#)  
*I2C FreeRTOS handle.* [More...](#)

#### I2C RTOS Operation

- status\_t [I2C\\_RRTOS\\_Init](#) (*i2c\_rtos\_handle\_t* \*handle, *I2C\_Type* \*base, const *i2c\_master\_config\_t* \*masterConfig, *uint32\_t* srcClock\_Hz)  
*Initializes I2C.*
- status\_t [I2C\\_RRTOS\\_Deinit](#) (*i2c\_rtos\_handle\_t* \*handle)  
*Deinitializes the I2C.*
- status\_t [I2C\\_RRTOS\\_Transfer](#) (*i2c\_rtos\_handle\_t* \*handle, *i2c\_master\_transfer\_t* \*transfer)  
*Performs I2C transfer.*

### 17.5.2 Data Structure Documentation

#### 17.5.2.1 struct i2c\_rtos\_handle\_t

##### Data Fields

- *I2C\_Type* \* **base**  
*I2C base address.*
- *i2c\_master\_handle\_t* **drv\_handle**  
*Handle of the underlying driver, treated as opaque by the RTOS layer.*
- *SemaphoreHandle\_t* **mutex**  
*Mutex to lock the handle during a transfer.*
- *SemaphoreHandle\_t* **sem**  
*Semaphore to notify and unblock task when transfer ends.*
- *OS\_EVENT* \* **mutex**  
*Mutex to lock the handle during a trasfer.*
- *OS\_FLAG\_GRP* \* **event**  
*Semaphore to notify and unblock task when transfer ends.*
- *OS\_SEM* **mutex**  
*Mutex to lock the handle during a trasfer.*
- *OS\_FLAG\_GRP* **event**  
*Semaphore to notify and unblock task when transfer ends.*

### 17.5.3 Function Documentation

**17.5.3.1 status\_t I2C\_RTOS\_Init ( i2c\_rtos\_handle\_t \* handle, I2C\_Type \* base, const i2c\_master\_config\_t \* masterConfig, uint32\_t srcClock\_Hz )**

This function initializes the I2C module and the related RTOS context.

Parameters

<i>handle</i>	The RTOS I2C handle, the pointer to an allocated space for RTOS context.
<i>base</i>	The pointer base address of the I2C instance to initialize.
<i>masterConfig</i>	Configuration structure to set-up I2C in master mode.
<i>srcClock_Hz</i>	Frequency of input clock of the I2C module.

Returns

status of the operation.

### 17.5.3.2 status\_t I2C\_RTOS\_Deinit ( i2c\_rtos\_handle\_t \* *handle* )

This function deinitializes the I2C module and the related RTOS context.

Parameters

<i>handle</i>	The RTOS I2C handle.
---------------	----------------------

### 17.5.3.3 status\_t I2C\_RTOS\_Transfer ( i2c\_rtos\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *transfer* )

This function performs an I2C transfer according to data given in the transfer structure.

Parameters

<i>handle</i>	The RTOS I2C handle.
<i>transfer</i>	Structure specifying the transfer parameters.

Returns

status of the operation.

### 17.6 I2C μCOS/II Driver

#### 17.6.1 Overview

#### Data Structures

- struct [i2c\\_rtos\\_handle\\_t](#)  
*I2C FreeRTOS handle.* [More...](#)

#### I2C RTOS Operation

- status\_t [I2C\\_RRTOS\\_Init](#) ([i2c\\_rtos\\_handle\\_t](#) \*handle, [I2C\\_Type](#) \*base, const [i2c\\_master\\_config\\_t](#) \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes I2C.*
- status\_t [I2C\\_RRTOS\\_Deinit](#) ([i2c\\_rtos\\_handle\\_t](#) \*handle)  
*Deinitializes the I2C.*
- status\_t [I2C\\_RRTOS\\_Transfer](#) ([i2c\\_rtos\\_handle\\_t](#) \*handle, [i2c\\_master\\_transfer\\_t](#) \*transfer)  
*Performs I2C transfer.*

#### 17.6.2 Data Structure Documentation

##### 17.6.2.1 struct i2c\_rtos\_handle\_t

#### Data Fields

- [I2C\\_Type](#) \* [base](#)  
*I2C base address.*
- [i2c\\_master\\_handle\\_t](#) [drv\\_handle](#)  
*Handle of the underlying driver, treated as opaque by the RTOS layer.*
- [SemaphoreHandle\\_t](#) [mutex](#)  
*Mutex to lock the handle during a transfer.*
- [SemaphoreHandle\\_t](#) [sem](#)  
*Semaphore to notify and unblock task when transfer ends.*
- [OS\\_EVENT](#) \* [mutex](#)  
*Mutex to lock the handle during a trasfer.*
- [OS\\_FLAG\\_GRP](#) \* [event](#)  
*Semaphore to notify and unblock task when transfer ends.*
- [OS\\_SEM](#) [mutex](#)  
*Mutex to lock the handle during a trasfer.*
- [OS\\_FLAG\\_GRP](#) [event](#)  
*Semaphore to notify and unblock task when transfer ends.*

### 17.6.3 Function Documentation

**17.6.3.1 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.

## I2C μCOS/II Driver

Parameters

<i>handle</i>	The RTOS I2C handle, the pointer to an allocated space for RTOS context.
<i>base</i>	The pointer base address of the I2C instance to initialize.
<i>masterConfig</i>	Configuration structure to set-up I2C in master mode.
<i>srcClock_Hz</i>	Frequency of input clock of the I2C module.

Returns

status of the operation.

### 17.6.3.2 status\_t I2C\_RTOS\_Deinit ( i2c\_rtos\_handle\_t \* *handle* )

This function deinitializes the I2C module and the related RTOS context.

Parameters

<i>handle</i>	The RTOS I2C handle.
---------------	----------------------

### 17.6.3.3 status\_t I2C\_RTOS\_Transfer ( i2c\_rtos\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *transfer* )

This function performs an I2C transfer according to data given in the transfer structure.

Parameters

<i>handle</i>	The RTOS I2C handle.
<i>transfer</i>	Structure specifying the transfer parameters.

Returns

status of the operation.

## 17.7 I2C µCOS/III Driver

### 17.7.1 Overview

#### Data Structures

- struct [i2c\\_rtos\\_handle\\_t](#)  
*I2C FreeRTOS handle.* [More...](#)

#### I2C RTOS Operation

- status\_t [I2C\\_RRTOS\\_Init](#) ([i2c\\_rtos\\_handle\\_t](#) \*handle, [I2C\\_Type](#) \*base, const [i2c\\_master\\_config\\_t](#) \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes I2C.*
- status\_t [I2C\\_RRTOS\\_Deinit](#) ([i2c\\_rtos\\_handle\\_t](#) \*handle)  
*Deinitializes the I2C.*
- status\_t [I2C\\_RRTOS\\_Transfer](#) ([i2c\\_rtos\\_handle\\_t](#) \*handle, [i2c\\_master\\_transfer\\_t](#) \*transfer)  
*Performs I2C transfer.*

### 17.7.2 Data Structure Documentation

#### 17.7.2.1 struct i2c\_rtos\_handle\_t

##### Data Fields

- [I2C\\_Type](#) \* [base](#)  
*I2C base address.*
- [i2c\\_master\\_handle\\_t](#) [drv\\_handle](#)  
*Handle of the underlying driver, treated as opaque by the RTOS layer.*
- [SemaphoreHandle\\_t](#) [mutex](#)  
*Mutex to lock the handle during a transfer.*
- [SemaphoreHandle\\_t](#) [sem](#)  
*Semaphore to notify and unblock task when transfer ends.*
- [OS\\_EVENT](#) \* [mutex](#)  
*Mutex to lock the handle during a trasfer.*
- [OS\\_FLAG\\_GRP](#) \* [event](#)  
*Semaphore to notify and unblock task when transfer ends.*
- [OS\\_SEM](#) [mutex](#)  
*Mutex to lock the handle during a trasfer.*
- [OS\\_FLAG\\_GRP](#) [event](#)  
*Semaphore to notify and unblock task when transfer ends.*

### 17.7.3 Function Documentation

**17.7.3.1 status\_t I2C\_RTOS\_Init ( i2c\_rtos\_handle\_t \* *handle*, I2C\_Type \* *base*, const i2c\_master\_config\_t \* *masterConfig*, uint32\_t *srcClock\_Hz* )**

This function initializes the I2C module and the related RTOS context.

Parameters

<i>handle</i>	The RTOS I2C handle, the pointer to an allocated space for RTOS context.
<i>base</i>	The pointer base address of the I2C instance to initialize.
<i>masterConfig</i>	Configuration structure to set-up I2C in master mode.
<i>srcClock_Hz</i>	Frequency of input clock of the I2C module.

Returns

status of the operation.

### 17.7.3.2 status\_t I2C\_RTOS\_Deinit ( i2c\_rtos\_handle\_t \* *handle* )

This function deinitializes the I2C module and the related RTOS context.

Parameters

<i>handle</i>	The RTOS I2C handle.
---------------	----------------------

### 17.7.3.3 status\_t I2C\_RTOS\_Transfer ( i2c\_rtos\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *transfer* )

This function performs an I2C transfer according to data given in the transfer structure.

Parameters

<i>handle</i>	The RTOS I2C handle.
<i>transfer</i>	Structure specifying the transfer parameters.

Returns

status of the operation.



# Chapter 18

## INTMUX: Interrupt Multiplexer Driver

### 18.1 Overview

The KSDK provides a peripheral driver for the Interrupt Multiplexer (INTMUX) module of Kinetis devices.

### 18.2 Typical use case

#### 18.2.1 Channel Configure

```
/* INTMUX initialization */
INTMUX_Init(INTMUX0);
/* Resets the INTMUX channel 0 */
INTMUX_ResetChannel(INTMUX0, 0);
/* Configures the INTMUX channel 0, enable INTMUX source 0, 1, OR mode. */
INTMUX_SetChannelConfig(INTMUX0, 0, 1<<0 | 1<<1, kINTMUX_ChannelLogicOR);
```

### Enumerations

- enum `intmux_channel_logic_mode_t` {  
    `kINTMUX_ChannelLogicOR` = 0x0U,  
    `kINTMUX_ChannelLogicAND` }  
*INTMUX channel logic mode.*

### Driver version

- #define `FSL_INTMUX_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 0)`)  
*< Version 2.0.0.*

### Initialization and deinitialization

- void `INTMUX_Init` (INTMUX\_Type \*base)  
*Initializes the INTMUX module.*
- void `INTMUX_Deinit` (INTMUX\_Type \*base)  
*Deinitializes an INTMUX instance for operation.*
- static void `INTMUX_ResetChannel` (INTMUX\_Type \*base, uint32\_t channel)  
*Resets an INTMUX channel.*
- static void `INTMUX_SetChannelMode` (INTMUX\_Type \*base, uint32\_t channel, `intmux_channel_logic_mode_t` logic)  
*Sets the logic mode for an INTMUX channel.*

### Sources

- static void `INTMUX_EnableInterrupt` (INTMUX\_Type \*base, uint32\_t channel, IRQn\_Type irq)  
*Enables an interrupt source on an INTMUX channel.*

## Function Documentation

- static void **INTMUX\_DisableInterrupt** (INTMUX\_Type \*base, uint32\_t channel, IRQn\_Type irq)  
*Disables an interrupt source on an INTMUX channel.*

## Status

- static uint32\_t **INTMUX\_GetChannelPendingSources** (INTMUX\_Type \*base, uint32\_t channel)  
*Gets INTMUX pending interrupt sources for a specific channel.*

## 18.3 Macro Definition Documentation

### 18.3.1 #define FSL\_INTMUX\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

## 18.4 Enumeration Type Documentation

### 18.4.1 enum intmux\_channel\_logic\_mode\_t

Enumerator

**kINTMUX\_ChannelLogicOR** Logic OR all enabled interrupt inputs.

**kINTMUX\_ChannelLogicAND** Logic AND all enabled interrupt inputs.

## 18.5 Function Documentation

### 18.5.1 void INTMUX\_Init ( INTMUX\_Type \* *base* )

This function enables the clock gate for the specified INTMUX. It then resets all channels, so that no interrupt sources are routed and the logic mode is set to default of **kINTMUX\_ChannelLogicOR**. Finally, the NVIC vectors for all the INTMUX output channels are enabled.

Parameters

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

### 18.5.2 void INTMUX\_Deinit ( INTMUX\_Type \* *base* )

The clock gate for the specified INTMUX is disabled and the NVIC vectors for all channels are disabled.

Parameters

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

### 18.5.3 static void INTMUX\_ResetChannel( INTMUX\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Sets all register values in the specified channel to their reset value. This function disables all interrupt sources for the channel.

## Function Documentation

Parameters

<i>base</i>	INTMUX peripheral base address.
<i>channel</i>	The INTMUX channel number.

### 18.5.4 static void INTMUX\_SetChannelMode ( INTMUX\_Type \* *base*, uint32\_t *channel*, intmux\_channel\_logic\_mode\_t *logic* ) [inline], [static]

INTMUX channels can be configured to use one of the two logic modes that control how pending interrupt sources on the channel trigger the output interrupt.

- [kINTMUX\\_ChannelLogicOR](#) means any source pending triggers the output interrupt.
- [kINTMUX\\_ChannelLogicAND](#) means all selected sources on the channel must be pending before the channel output interrupt triggers.

Parameters

<i>base</i>	INTMUX peripheral base address.
<i>channel</i>	The INTMUX channel number.
<i>logic</i>	The INTMUX channel logic mode.

### 18.5.5 static void INTMUX\_EnableInterrupt ( INTMUX\_Type \* *base*, uint32\_t *channel*, IRQn\_Type *irq* ) [inline], [static]

Parameters

<i>base</i>	INTMUX peripheral base address.
<i>channel</i>	Index of the INTMUX channel on which the specified interrupt is enabled.
<i>irq</i>	Interrupt to route to the specified INTMUX channel. The interrupt must be an INT-MUX source.

### 18.5.6 static void INTMUX\_DisableInterrupt ( INTMUX\_Type \* *base*, uint32\_t *channel*, IRQn\_Type *irq* ) [inline], [static]

Parameters

<i>base</i>	INTMUX peripheral base address.
<i>channel</i>	Index of the INTMUX channel on which the specified interrupt is disabled.
<i>irq</i>	Interrupt number. The interrupt must be an INTMUX source.

### 18.5.7 static uint32\_t INTMUX\_GetChannelPendingSources ( INTMUX\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters

<i>base</i>	INTMUX peripheral base address.
<i>channel</i>	The INTMUX channel number.

Returns

The mask of pending interrupt bits. Bit[n] set means INTMUX source n is pending.

## Function Documentation

# Chapter 19

## LLWU: Low-Leakage Wakeup Unit Driver

### 19.1 Overview

The KSDK provides a Peripheral driver for the Low-Leakage Wakeup Unit (LLWU) module of Kinetis devices. The LLWU module allows the user to select external pin sources and internal modules as a wake-up source from low-leakage power modes.

### 19.2 External wakeup pins configurations

Configures the external wakeup pins' working modes, gets and clears the wake pin flags. External wakeup pins are accessed by `pinIndex` which is started from 1. Numbers of external pins depend on the SoC configuration.

### 19.3 Internal wakeup modules configurations

Enables/disables the internal wakeup modules, and gets the modules flags. Internal modules are accessed by `moduleIndex` which is started from 1. Numbers of external pins depend the on SoC configuration.

### 19.4 Digital pin filter for external wakeup pin configurations

Configures the digital pin filter of the external wakeup pins' working modes, gets and clears the pin filter flags. Digital pins filters are accessed by `filterIndex` which is started from 1. Numbers of external pins depends on the SoC configuration.

## Data Structures

- struct `llwu_external_pin_filter_mode_t`  
*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.*

## Enumeration Type Documentation

### Driver version

- #define **FSL\_LLWU\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 1))  
*LLWU driver version 2.0.1.*

### Low-Leakage Wakeup Unit Control APIs

- void **LLWU\_SetExternalWakeUpPinMode** (LLWU\_Type \*base, uint32\_t pinIndex, **llwu\_external\_pin\_mode\_t** pinMode)  
*Sets the external input pin source mode.*
- bool **LLWU\_GetExternalWakeUpPinFlag** (LLWU\_Type \*base, uint32\_t pinIndex)  
*Gets the external wakeup source flag.*
- void **LLWU\_ClearExternalWakeUpPinFlag** (LLWU\_Type \*base, uint32\_t pinIndex)  
*Clears the external wakeup source flag.*
- static void **LLWU\_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.*
- 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)  
*Clear the pin filter configuration.*

## 19.5 Data Structure Documentation

### 19.5.1 struct **llwu\_external\_pin\_filter\_mode\_t**

#### Data Fields

- uint32\_t **pinIndex**  
*Pin number.*
- **llwu\_pin\_filter\_mode\_t** **filterMode**  
*Filter mode.*

## 19.6 Macro Definition Documentation

### 19.6.1 #define **FSL\_LLWU\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 1))

## 19.7 Enumeration Type Documentation

### 19.7.1 enum **llwu\_external\_pin\_mode\_t**

Enumerator

**kLLWU\_ExternalPinDisable** Pin disabled as wakeup input.

*kLLWU\_ExternalPinRisingEdge* Pin enabled with rising edge detection.

*kLLWU\_ExternalPinFallingEdge* Pin enabled with falling edge detection.

*kLLWU\_ExternalPinAnyEdge* Pin enabled with any change detection.

## 19.7.2 enum llwu\_pin\_filter\_mode\_t

Enumerator

*kLLWU\_PinFilterDisable* Filter disabled.

*kLLWU\_PinFilterRisingEdge* Filter positive edge detection.

*kLLWU\_PinFilterFallingEdge* Filter negative edge detection.

*kLLWU\_PinFilterAnyEdge* Filter any edge detection.

## 19.8 Function Documentation

### 19.8.1 void LLWU\_SetExternalWakeupsPinMode ( *LLWU\_Type \* base*, *uint32\_t pinIndex*, *llwu\_external\_pin\_mode\_t pinMode* )

This function sets the external input pin source mode that is used as a wake up source.

Parameters

<i>base</i>	LLWU peripheral base address.
<i>pinIndex</i>	pin index which to be enabled as external wakeup source, start from 1.
<i>pinMode</i>	pin configuration mode defined in llwu_external_pin_modes_t

### 19.8.2 bool LLWU\_GetExternalWakeupsPinFlag ( *LLWU\_Type \* base*, *uint32\_t pinIndex* )

This function checks the external pin flag to detect whether the MCU is woke up by the specific pin.

Parameters

<i>base</i>	LLWU peripheral base address.
<i>pinIndex</i>	pin index, start from 1.

Returns

true if the specific pin is wake up source.

## Function Documentation

**19.8.3 void LLWU\_ClearExternalWakeupPinFlag ( *LLWU\_Type* \* *base*, *uint32\_t* *pinIndex* )**

This function clears the external wakeup source flag for a specific pin.

Parameters

<i>base</i>	LLWU peripheral base address.
<i>pinIndex</i>	pin index, start from 1.

#### 19.8.4 static void LLWU\_EnableInternalModuleInterruptWakup ( LLWU\_Type \* *base*, uint32\_t *moduleIdx*, 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>moduleIdx</i>	module index which to be enabled as internal wakeup source, start from 1.
<i>enable</i>	enable or disable setting

#### 19.8.5 static bool LLWU\_GetInternalWakeupModuleFlag ( LLWU\_Type \* *base*, uint32\_t *moduleIdx* ) [inline], [static]

This function checks the external pin flag to detect whether the system is woke up by the specific pin.

Parameters

<i>base</i>	LLWU peripheral base address.
<i>moduleIdx</i>	module index, start from 1.

Returns

true if the specific pin is wake up source.

#### 19.8.6 void LLWU\_SetPinFilterMode ( LLWU\_Type \* *base*, uint32\_t *filterIndex*, llwu\_external\_pin\_filter\_mode\_t *filterMode* )

This function sets the pin filter configuration.

## Function Documentation

Parameters

<i>base</i>	LLWU peripheral base address.
<i>filterIndex</i>	pin filter index which used to enable/disable the digital filter, start from 1.
<i>filterMode</i>	filter mode configuration

### **19.8.7 bool LLWU\_GetPinFilterFlag ( LLWU\_Type \* *base*, uint32\_t *filterIndex* )**

This function gets the pin filter flag.

Parameters

<i>base</i>	LLWU peripheral base address.
<i>filterIndex</i>	pin filter index, start from 1.

Returns

true if the flag is a source of existing a low-leakage power mode.

### **19.8.8 void LLWU\_ClearPinFilterFlag ( LLWU\_Type \* *base*, uint32\_t *filterIndex* )**

This function clear the pin filter flag.

Parameters

<i>base</i>	LLWU peripheral base address.
<i>filterIndex</i>	pin filter index which to be clear the flag, start from 1.

# Chapter 20

## LPTMR: Low-Power Timer

### 20.1 Overview

The KSDK provides a driver for the Low-Power Timer (LPTMR) of Kinetis devices.

### 20.2 Function groups

The LPTMR driver supports operating the module as a time counter or as a pulse counter.

#### 20.2.1 Initialization and deinitialization

The function [LPTMR\\_Init\(\)](#) initializes the LPTMR with specified configurations. The function [LPTMR\\_GetDefaultConfig\(\)](#) gets the default configurations. The initialization function configures the LPTMR for timer or pulse counter mode mode. It also sets up the LPTMR's free running mode operation and clock source.

The function [LPTMR\\_DeInit\(\)](#) disables the LPTMR module and gate the module clock.

#### 20.2.2 Timer period Operations

The function [LPTMR\\_SetTimerPeriod\(\)](#) sets the timer period in units of count. Timers counts from 0 till it equals the count value set here.

The function [LPTMR\\_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. User can call the utility macros provided in `fsl_common.h` to convert to microseconds or milliseconds

#### 20.2.3 Start and Stop timer operations

The function [LPTMR\\_StartTimer\(\)](#) starts the timer counting. After calling this function, the timer counts up to the count value set earlier via the [LPTMR\\_SetPeriod\(\)](#) function. Each time the timer reaches count 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.

The function [LPTMR\\_StopTimer\(\)](#) stops the timer counting and resets the timer's counter register

## Typical use case

### 20.2.4 Status

Provides functions to get and clear the LPTMR status.

### 20.2.5 Interrupt

Provides functions to enable/disable LPTMR interrupts and get current enabled interrupts.

## 20.3 Typical use case

### 20.3.1 LPTMR tick example

Updates the LPTMR period and toggles an LED periodically.

```
int main(void)
{
    uint32_t currentCounter = 0U;
    lptmr_config_t lptmrConfig;

    LED_INIT();

    /* Board pin, clock, debug console init */
    BOARD_InitHardware();

    /* Configure LPTMR */
    LPTMR_GetDefaultConfig(&lptmrConfig);

    /* Initialize the LPTMR */
    LPTMR_Init(LPTMRO, &lptmrConfig);

    /* Set timer period */
    LPTMR_SetTimerPeriod(LPTMRO, USEC_TO_COUNT(1000000U, LPTMR_SOURCE_CLOCK));

    /* Enable timer interrupt */
    LPTMR_EnableInterrupts(LPTMRO,
                           kLPTMR_TimerInterruptEnable);

    /* Enable at the NVIC */
    EnableIRQ(LPTMRO_IRQn);

    PRINTF("Low Power Timer Example\r\n");

    /* Start counting */
    LPTMR_StartTimer(LPTMRO);
    while (1)
    {
        if (currentCounter != lptmrCounter)
        {
            currentCounter = lptmrCounter;
            PRINTF("LPTMR interrupt No.%d \r\n", currentCounter);
        }
    }
}
```

## 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 LPTMR interrupts.*
- enum `lptmr_status_flags_t` { `kLPTMR_TimerCompareFlag` = LPTMR\_CSR\_TCF\_MASK }
   
*List of LPTMR status flags.*

## Driver version

- #define `FSL_LPTMR_DRIVER_VERSION` (`MAKE_VERSION`(2, 0, 0))

## Data Structure Documentation

Version 2.0.0.

### Initialization and deinitialization

- void [LPTMR\\_Init](#) (LPTMR\_Type \*base, const [lptmr\\_config\\_t](#) \*config)  
*Ungate the LPTMR clock and configures the peripheral for basic operation.*
- void [LPTMR\\_Deinit](#) (LPTMR\_Type \*base)  
*Gate the LPTMR clock.*
- void [LPTMR\\_GetDefaultConfig](#) ([lptmr\\_config\\_t](#) \*config)  
*Fill in the LPTMR config struct with the 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, uint16\_t ticks)  
*Sets the timer period in units of count.*
- static uint16\_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 counting.*
- static void [LPTMR\\_StopTimer](#) (LPTMR\_Type \*base)  
*Stops the timer counting.*

## 20.4 Data Structure Documentation

### 20.4.1 `struct lptmr_config_t`

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the [LPTMR\\_GetDefaultConfig\(\)](#) function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

## Data Fields

- [lptmr\\_timer\\_mode\\_t timerMode](#)  
*Time counter mode or pulse counter mode.*
- [lptmr\\_pin\\_select\\_t pinSelect](#)  
*LPTMR pulse input pin select; used only in pulse counter mode.*
- [lptmr\\_pin\\_polarity\\_t pinPolarity](#)  
*LPTMR pulse input pin polarity; used only in pulse counter mode.*
- [bool enableFreeRunning](#)  
*true: enable free running, counter is reset on overflow false: counter is reset when the compare flag is set*
- [bool bypassPrescaler](#)  
*true: bypass prescaler; false: use clock from prescaler*
- [lptmr\\_prescaler\\_clock\\_select\\_t prescalerClockSource](#)  
*LPTMR clock source.*
- [lptmr\\_prescaler\\_glitch\\_value\\_t value](#)  
*Prescaler or glitch filter value.*

## 20.5 Enumeration Type Documentation

### 20.5.1 enum lptmr\_pin\_select\_t

Enumerator

- kLPTMR\_PinSelectInput\_0*** Pulse counter input 0 is selected.
- kLPTMR\_PinSelectInput\_1*** Pulse counter input 1 is selected.
- kLPTMR\_PinSelectInput\_2*** Pulse counter input 2 is selected.
- kLPTMR\_PinSelectInput\_3*** Pulse counter input 3 is selected.

### 20.5.2 enum lptmr\_pin\_polarity\_t

Enumerator

- kLPTMR\_PinPolarityActiveHigh*** Pulse Counter input source is active-high.
- kLPTMR\_PinPolarityActiveLow*** Pulse Counter input source is active-low.

### 20.5.3 enum lptmr\_timer\_mode\_t

Enumerator

- kLPTMR\_TimerModeTimeCounter*** Time Counter mode.
- kLPTMR\_TimerModePulseCounter*** Pulse Counter mode.

## Enumeration Type Documentation

### 20.5.4 enum lptmr\_prescaler\_glitch\_value\_t

Enumerator

- kLPTMR\_Prescale\_Glitch\_0*** Prescaler divide 2, glitch filter does not support this setting.
- kLPTMR\_Prescale\_Glitch\_1*** Prescaler divide 4, glitch filter 2.
- kLPTMR\_Prescale\_Glitch\_2*** Prescaler divide 8, glitch filter 4.
- kLPTMR\_Prescale\_Glitch\_3*** Prescaler divide 16, glitch filter 8.
- kLPTMR\_Prescale\_Glitch\_4*** Prescaler divide 32, glitch filter 16.
- kLPTMR\_Prescale\_Glitch\_5*** Prescaler divide 64, glitch filter 32.
- kLPTMR\_Prescale\_Glitch\_6*** Prescaler divide 128, glitch filter 64.
- kLPTMR\_Prescale\_Glitch\_7*** Prescaler divide 256, glitch filter 128.
- kLPTMR\_Prescale\_Glitch\_8*** Prescaler divide 512, glitch filter 256.
- kLPTMR\_Prescale\_Glitch\_9*** Prescaler divide 1024, glitch filter 512.
- kLPTMR\_Prescale\_Glitch\_10*** Prescaler divide 2048 glitch filter 1024.
- kLPTMR\_Prescale\_Glitch\_11*** Prescaler divide 4096, glitch filter 2048.
- kLPTMR\_Prescale\_Glitch\_12*** Prescaler divide 8192, glitch filter 4096.
- kLPTMR\_Prescale\_Glitch\_13*** Prescaler divide 16384, glitch filter 8192.
- kLPTMR\_Prescale\_Glitch\_14*** Prescaler divide 32768, glitch filter 16384.
- kLPTMR\_Prescale\_Glitch\_15*** Prescaler divide 65536, glitch filter 32768.

### 20.5.5 enum lptmr\_prescaler\_clock\_select\_t

Note

Clock connections are SoC-specific

Enumerator

- kLPTMR\_PrescalerClock\_0*** Prescaler/glitch filter clock 0 selected.
- kLPTMR\_PrescalerClock\_1*** Prescaler/glitch filter clock 1 selected.
- kLPTMR\_PrescalerClock\_2*** Prescaler/glitch filter clock 2 selected.
- kLPTMR\_PrescalerClock\_3*** Prescaler/glitch filter clock 3 selected.

### 20.5.6 enum lptmr\_interrupt\_enable\_t

Enumerator

- kLPTMR\_TimerInterruptEnable*** Timer interrupt enable.

## 20.5.7 enum lptmr\_status\_flags\_t

Enumerator

*kLPTMR\_TimerCompareFlag* Timer compare flag.

## 20.6 Function Documentation

### 20.6.1 void LPTMR\_Init ( LPTMR\_Type \* *base*, const lptmr\_config\_t \* *config* )

Note

This API should be called at the beginning of the application using the LPTMR driver.

Parameters

<i>base</i>	LPTMR peripheral base address
<i>config</i>	Pointer to user's LPTMR config structure.

### 20.6.2 void LPTMR\_Deinit ( LPTMR\_Type \* *base* )

Parameters

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

### 20.6.3 void LPTMR\_GetDefaultConfig ( lptmr\_config\_t \* *config* )

The default values are:

```
*     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

## Function Documentation

<i>config</i>	Pointer to user's LPTMR config structure.
---------------	---

### 20.6.4 static void LPTMR\_EnableInterrupts ( LPTMR\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	LPTMR peripheral base address
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">lptmr_interrupt_enable_t</a>

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

Parameters

<i>base</i>	LPTMR peripheral base address
<i>mask</i>	The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">lptmr_interrupt_enable_t</a>

### 20.6.6 static uint32\_t LPTMR\_GetEnabledInterrupts ( LPTMR\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [lptmr\\_interrupt\\_enable\\_t](#)

### 20.6.7 static uint32\_t LPTMR\_GetStatusFlags ( LPTMR\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

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

#### 20.6.8 static void LPTMR\_ClearStatusFlags ( LPTMR\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

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

#### 20.6.9 static void LPTMR\_SetTimerPeriod ( LPTMR\_Type \* *base*, uint16\_t *ticks* ) [inline], [static]

Timers counts from 0 till 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. User can call the utility macros provided in `fsl_common.h` to convert to ticks

Parameters

<i>base</i>	LPTMR peripheral base address
<i>ticks</i>	Timer period in units of ticks

#### 20.6.10 static uint16\_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

User can call the utility macros provided in `fsl_common.h` to convert ticks to usec or msec

## Function Documentation

Parameters

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

Returns

Current counter value in ticks

### 20.6.11 static void LPTMR\_StartTimer ( LPTMR\_Type \* *base* ) [inline], [static]

After calling this function, the timer counts up to the CMR register value. Each time the timer reaches CMR value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

Parameters

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

### 20.6.12 static void LPTMR\_StopTimer ( LPTMR\_Type \* *base* ) [inline], [static]

This function stops the timer counting and resets the timer's counter register

Parameters

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

# Chapter 21

## LPUART: Low Power UART Driver

### 21.1 Overview

#### Modules

- LPUART DMA Driver
- LPUART Driver
- LPUART FreeRTOS Driver
- LPUART eDMA Driver
- LPUART µCOS/II Driver
- LPUART µCOS/III Driver

## LPUART Driver

### 21.2 LPUART Driver

#### 21.2.1 Overview

The KSDK provides a peripheral driver for the Low Power UART (LPUART) module of Kinetis devices.

#### 21.2.2 Typical use case

##### 21.2.2.1 LPUART Operation

```
uint8_t ch;
LPUART_GetDefaultConfig(&user_config);
user_config.baudRate = 115200U;

LPUART_Init(LPUART1, &user_config, 120000000U);

LPUART_SendDataPolling(LPUART1, txbuff, sizeof(txbuff));

while(1)
{
    LPUART_ReceiveDataPolling(LPUART1, &ch, 1);
    LPUART_SendDataPolling(LPUART1, &ch, 1);
}
```

## Data Structures

- struct [lpuart\\_config\\_t](#)  
*LPUART configure structure.* [More...](#)
- struct [lpuart\\_transfer\\_t](#)  
*LPUART transfer structure.* [More...](#)
- struct [lpuart\\_handle\\_t](#)  
*LPUART handle structure.* [More...](#)

## 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 `_lpuart_status` {
   
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,
   
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 }
   
*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_interrupt_enable` {
   
kLPUART\_RxActiveEdgeInterruptEnable = (LPUART\_BAUD\_RXEDGIE\_MASK >> 8),
   
kLPUART\_TxDataRegEmptyInterruptEnable = (LPUART\_CTRL\_TIE\_MASK),
   
kLPUART\_TransmissionCompleteInterruptEnable = (LPUART\_CTRL\_TCIE\_MASK),
   
kLPUART\_RxDataRegFullInterruptEnable = (LPUART\_CTRL\_RIE\_MASK),
   
kLPUART\_IdleLineInterruptEnable = (LPUART\_CTRL\_ILIE\_MASK),
   
kLPUART\_RxOverrunInterruptEnable = (LPUART\_CTRL\_ORIE\_MASK),
   
kLPUART\_NoiseErrorInterruptEnable = (LPUART\_CTRL\_NEIE\_MASK),
   
kLPUART\_FramingErrorInterruptEnable = (LPUART\_CTRL\_FEIE\_MASK),
   
kLPUART\_ParityErrorInterruptEnable = (LPUART\_CTRL\_PEIE\_MASK) }
   
*LPUART interrupt configuration structure, default settings all disabled.*
- enum `_lpuart_flags` {

## LPUART Driver

```
kLPUART_TxDataRegEmptyFlag,  
kLPUART_TransmissionCompleteFlag,  
kLPUART_RxDataRegFullFlag,  
kLPUART_IdleLineFlag = (LPUART_STAT_IDLE_MASK),  
kLPUART_RxOverrunFlag = (LPUART_STAT_OR_MASK),  
kLPUART_NoiseErrorFlag = (LPUART_STAT_NF_MASK),  
kLPUART_FramingErrorFlag,  
kLPUART_ParityErrorFlag = (LPUART_STAT_PF_MASK),  
kLPUART_RxActiveEdgeFlag,  
kLPUART_RxActiveFlag }
```

*LPUART status flags.*

## Driver version

- #define **FSL\_LPUART\_DRIVER\_VERSION** (MAKE\_VERSION(2, 2, 1))  
*LPUART driver version 2.2.1.*

## 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.*
- status\_t **LPUART\_SetBaudRate** (LPUART\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the LPUART instance baudrate.*

## Status

- uint32\_t **LPUART\_GetStatusFlags** (LPUART\_Type \*base)  
*Gets LPUART status flags.*
- status\_t **LPUART\_ClearStatusFlags** (LPUART\_Type \*base, uint32\_t mask)  
*Clears status flags with a provided mask.*

## Interrupts

- void **LPUART\_EnableInterrupts** (LPUART\_Type \*base, uint32\_t mask)  
*Enables LPUART interrupts according to a provided mask.*
- void **LPUART\_DisableInterrupts** (LPUART\_Type \*base, uint32\_t mask)  
*Disables LPUART interrupts according to a provided mask.*
- uint32\_t **LPUART\_GetEnabledInterrupts** (LPUART\_Type \*base)

*Gets enabled LPUART interrupts.*

## Bus Operations

- 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 RX register.*
- void **LPUART\_WriteBlocking** (LPUART\_Type \*base, const uint8\_t \*data, size\_t length)  
*Writes to transmitter register using a blocking method.*
- status\_t **LPUART\_ReadBlocking** (LPUART\_Type \*base, uint8\_t \*data, size\_t length)  
*Reads the RX 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)  
*Abort the background transfer and uninstall the 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)  
*Get the number of bytes that have been written to LPUART TX register.*
- 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)  
*Get the number of bytes that have been received.*
- void **LPUART\_TransferHandleIRQ** (LPUART\_Type \*base, lpuart\_handle\_t \*handle)  
*LPUART IRQ handle function.*
- void **LPUART\_TransferHandleErrorIRQ** (LPUART\_Type \*base, lpuart\_handle\_t \*handle)  
*LPUART Error IRQ handle function.*

### 21.2.3 Data Structure Documentation

#### 21.2.3.1 struct lpuart\_config\_t

##### Data Fields

- uint32\_t **baudRate\_Bps**  
*LPUART baud rate.*
- **lpuart\_parity\_mode\_t parityMode**  
*Parity mode, disabled (default), even, odd.*
- **lpuart\_data\_bits\_t dataBitsCount**  
*Data bits count, eight (default), seven.*
- bool **isMsb**  
*Data bits order, LSB (default), MSB.*
- bool **enableTx**  
*Enable TX.*
- bool **enableRx**  
*Enable RX.*

#### 21.2.3.2 struct lpuart\_transfer\_t

##### Data Fields

- uint8\_t \* **data**  
*The buffer of data to be transfer.*
- size\_t **dataSize**  
*The byte count to be transfer.*

#### 21.2.3.2.0.9 Field Documentation

##### 21.2.3.2.0.9.1 uint8\_t\* lpuart\_transfer\_t::data

##### 21.2.3.2.0.9.2 size\_t lpuart\_transfer\_t::dataSize

#### 21.2.3.3 struct \_lpuart\_handle

##### Data Fields

- uint8\_t \*volatile **txData**  
*Address of remaining data to send.*
- volatile size\_t **txDataSize**  
*Size of the remaining data to send.*
- size\_t **txDataSizeAll**  
*Size of the data to send out.*
- uint8\_t \*volatile **rxData**  
*Address of remaining data to receive.*
- volatile size\_t **rxDataSize**  
*Size of the remaining data to receive.*
- size\_t **rxDataSizeAll**

- **uint8\_t \* rxRingBuffer**  
*Size of the data to receive.*
- **size\_t rxRingBufferSize**  
*Start address of the receiver ring buffer.*
- **volatile uint16\_t rxRingBufferHead**  
*Size of the ring buffer.*
- **volatile uint16\_t rxRingBufferTail**  
*Index for the driver to store received data into ring buffer.*
- **lpuart\_transfer\_callback\_t callback**  
*Index for the user to get data from the ring buffer.*
- **void \* userData**  
*LPUART callback function parameter.*
- **volatile uint8\_t txState**  
*TX transfer state.*
- **volatile uint8\_t rxState**  
*RX transfer state.*

## LPUART Driver

### 21.2.3.3.0.10 Field Documentation

- 21.2.3.3.0.10.1 `uint8_t* volatile Ipuart_handle_t::txData`
- 21.2.3.3.0.10.2 `volatile size_t Ipuart_handle_t::txDataSize`
- 21.2.3.3.0.10.3 `size_t Ipuart_handle_t::txDataSizeAll`
- 21.2.3.3.0.10.4 `uint8_t* volatile Ipuart_handle_t::rxData`
- 21.2.3.3.0.10.5 `volatile size_t Ipuart_handle_t::rxDataSize`
- 21.2.3.3.0.10.6 `size_t Ipuart_handle_t::rxDataSizeAll`
- 21.2.3.3.0.10.7 `uint8_t* Ipuart_handle_t::rxRingBuffer`
- 21.2.3.3.0.10.8 `size_t Ipuart_handle_t::rxRingBufferSize`
- 21.2.3.3.0.10.9 `volatile uint16_t Ipuart_handle_t::rxRingBufferHead`
- 21.2.3.3.0.10.10 `volatile uint16_t Ipuart_handle_t::rxRingBufferTail`
- 21.2.3.3.0.10.11 `Ipuart_transfer_callback_t Ipuart_handle_t::callback`
- 21.2.3.3.0.10.12 `void* Ipuart_handle_t::userData`
- 21.2.3.3.0.10.13 `volatile uint8_t Ipuart_handle_t::txState`
- 21.2.3.3.0.10.14 `volatile uint8_t Ipuart_handle_t::rxState`

### 21.2.4 Macro Definition Documentation

- 21.2.4.1 `#define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))`

### 21.2.5 Typedef Documentation

- 21.2.5.1 `typedef void(* Ipuart_transfer_callback_t)(LPUART_Type *base, Ipuart_handle_t *handle, status_t status, void *userData)`

### 21.2.6 Enumeration Type Documentation

#### 21.2.6.1 enum \_Ipuart\_status

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.

### 21.2.6.2 enum lpuart\_parity\_mode\_t

Enumerator

*kLPUART\_ParityDisabled* Parity disabled.  
*kLPUART\_ParityEven* Parity enabled, type even, bit setting: PE|PT = 10.  
*kLPUART\_ParityOdd* Parity enabled, type odd, bit setting: PE|PT = 11.

### 21.2.6.3 enum lpuart\_data\_bits\_t

Enumerator

*kLPUART\_EightDataBits* Eight data bit.

### 21.2.6.4 enum lpuart\_stop\_bit\_count\_t

Enumerator

*kLPUART\_OneStopBit* One stop bit.  
*kLPUART\_TwoStopBit* Two stop bits.

### 21.2.6.5 enum \_lpuart\_interrupt\_enable

This structure contains the settings for all LPUART interrupt configurations.

Enumerator

*kLPUART\_RxActiveEdgeInterruptEnable* Receive Active Edge.  
*kLPUART\_TxDataRegEmptyInterruptEnable* Transmit data register empty.  
*kLPUART\_TransmissionCompleteInterruptEnable* Transmission complete.

## LPUART Driver

***kLPUART\_RxDataRegFullInterruptEnable*** Receiver data register full.

***kLPUART\_IdleLineInterruptEnable*** Idle line.

***kLPUART\_RxOverrunInterruptEnable*** Receiver Overrun.

***kLPUART\_NoiseErrorInterruptEnable*** Noise error flag.

***kLPUART\_FramingErrorInterruptEnable*** Framing error flag.

***kLPUART\_ParityErrorInterruptEnable*** Parity error flag.

### 21.2.6.6 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.

***kLPUART\_TransmissionCompleteFlag*** Transmission complete flag, sets when transmission activity complete.

***kLPUART\_RxDataRegFullFlag*** Receive data register full flag, sets when the receive data buffer is full.

***kLPUART\_IdleLineFlag*** Idle line detect flag, sets when idle line detected.

***kLPUART\_RxOverrunFlag*** Receive Overrun, sets when new data is received before data is read from receive register.

***kLPUART\_NoiseErrorFlag*** Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets

***kLPUART\_FramingErrorFlag*** Frame error flag, sets if logic 0 was detected where stop bit expected.

***kLPUART\_ParityErrorFlag*** If parity enabled, sets upon parity error detection.

***kLPUART\_RxActiveEdgeFlag*** Receive pin active edge interrupt flag, sets when active edge detected.

***kLPUART\_RxActiveFlag*** Receiver Active Flag (RAF), sets at beginning of valid start bit.

### 21.2.7 Function Documentation

#### 21.2.7.1 **status\_t LPUART\_Init ( LPUART\_Type \* base, const lpuart\_config\_t \* config, uint32\_t srcClock\_Hz )**

This function configures the LPUART module with user-defined settings. Call the [LPUART\\_GetDefaultConfig\(\)](#) function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
* lpuart_config_t lpuartConfig;
* lpuartConfig.baudRate_Bps = 115200U;
* lpuartConfig.parityMode = kLPUART_ParityDisabled;
* lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
* lpuartConfig.isMsb = false;
```

```

* lpuartConfig.stopBitCount = kLPUART_OneStopBit;
* lpuartConfig.txFifoWatermark = 0;
* lpuartConfig.rxFifoWatermark = 1;
* LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
*

```

#### Parameters

<i>base</i>	LPUART peripheral base address.
<i>config</i>	Pointer to a user-defined configuration structure.
<i>srcClock_Hz</i>	LPUART clock source frequency in HZ.

#### Return values

<i>kStatus_LPUART_BaudrateNotSupport</i>	Baudrate is not support in current clock source.
<i>kStatus_Success</i>	LPUART initialize succeed

### 21.2.7.2 void LPUART\_Deinit( LPUART\_Type \* *base* )

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

#### Parameters

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

### 21.2.7.3 void LPUART\_GetDefaultConfig( lpuart\_config\_t \* *config* )

This function initializes the LPUART configuration structure to a default value. The default values are:  
: lpuartConfig->baudRate\_Bps = 115200U; lpuartConfig->parityMode = kLPUART\_ParityDisabled;  
lpuartConfig->dataBitsCount = kLPUART\_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART\_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

#### Parameters

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

### 21.2.7.4 status\_t LPUART\_SetBaudRate( LPUART\_Type \* *base*, uint32\_t *baudRate\_Bps*, uint32\_t *srcClock\_Hz* )

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART\_Init.

## LPUART Driver

```
* LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);  
*
```

Parameters

<i>base</i>	LPUART peripheral base address.
<i>baudRate_Bps</i>	LPUART baudrate to be set.
<i>srcClock_Hz</i>	LPUART clock source frequency in HZ.

Return values

<i>kStatus_LPUART_BaudrateNotSupport</i>	Baudrate is not supported in the current clock source.
<i>kStatus_Success</i>	Set baudrate succeeded.

### 21.2.7.5 `uint32_t LPUART_GetStatusFlags( LPUART_Type * base )`

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators `_lpuart_flags`. To check for a specific status, compare the return value with enumerators in the `_lpuart_flags`. For example, to check whether the TX is empty:

```
* if (kLPUART_TxDataRegEmptyFlag &  
     LPUART_GetStatusFlags(LPUART1))  
* {  
*     ...  
* }  
*
```

Parameters

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

Returns

LPUART status flags which are ORed by the enumerators in the `_lpuart_flags`.

### 21.2.7.6 `status_t LPUART_ClearStatusFlags( LPUART_Type * base, uint32_t mask )`

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only be cleared or set by hardware are: kLPUART\_TxDataRegEmptyFlag, kLPUART\_TransmissionCompleteFlag, kLPUART\_RxDataRegFullFlag, kLPUART\_RxActiveFlag, kLPUART\_NoiseErrorInRxDataRegFlag, kLPUART\_ParityErrorInRxDataRegFlag, kLPUART\_TxFifoEmptyFlag, kLPUART\_RxFifoEmptyFlag. Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

## Parameters

<i>base</i>	LPUART peripheral base address.
<i>mask</i>	the status flags to be cleared. The user can use the enumerators in the <code>_lpuart_status_flag_t</code> to do the OR operation and get the mask.

## Returns

0 succeed, others failed.

## Return values

<i>kStatus_LPUART_Flag_CannotClearManually</i>	The flag can't be cleared by this function but it is cleared automatically by hardware.
<i>kStatus_Success</i>	Status in the mask are cleared.

**21.2.7.7 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 <code>_uart_interrupt_enable</code> .

**21.2.7.8 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);
*
```

## LPUART Driver

Parameters

<i>base</i>	LPUART peripheral base address.
<i>mask</i>	The interrupts to disable. Logical OR of <a href="#">_lpuart_interrupt_enable</a> .

### 21.2.7.9 **uint32\_t LPUART\_GetEnabledInterrupts ( LPUART\_Type \* *base* )**

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators [\\_lpuart\\_interrupt\\_enable](#). To check a specific interrupt enable status, compare the return value with enumerators in [\\_lpuart\\_interrupt\\_enable](#). For example, to check whether the TX empty interrupt is enabled:

```
*     uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);
*
*     if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
*     {
*         ...
*     }
*
```

Parameters

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

Returns

LPUART interrupt flags which are logical OR of the enumerators in [\\_lpuart\\_interrupt\\_enable](#).

### 21.2.7.10 **static void LPUART\_EnableTx ( LPUART\_Type \* *base*, bool *enable* ) [inline], [static]**

This function enables or disables the LPUART transmitter.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>enable</i>	True to enable, false to disable.

### 21.2.7.11 **static void LPUART\_EnableRx ( LPUART\_Type \* *base*, bool *enable* ) [inline], [static]**

This function enables or disables the LPUART receiver.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>enable</i>	True to enable, false to disable.

#### **21.2.7.12 static void LPUART\_WriteByte ( LPUART\_Type \* *base*, uint8\_t *data* ) [inline], [static]**

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>data</i>	Data write to the TX register.

#### **21.2.7.13 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 RX register is full or that the RX FIFO has data before calling this function.

Parameters

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

Returns

Data read from data register.

#### **21.2.7.14 void LPUART\_WriteBlocking ( LPUART\_Type \* *base*, const uint8\_t \* *data*, size\_t *length* )**

This function polls the transmitter register, waits for the register to be empty or for TX FIFO to have room and then writes data to the transmitter buffer.

Note

This function does not check whether all data has been sent out to the bus. Before disabling the transmitter, check the kLPUART\_TransmissionCompleteFlag to ensure that the transmit is finished.

## LPUART Driver

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.

### 21.2.7.15 **status\_t LPUART\_ReadBlocking ( LPUART\_Type \* *base*, uint8\_t \* *data*, size\_t *length* )**

This function polls the RX register, waits for the RX register full or RX FIFO has data then 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_Success</i>	Successfully received all data.

### 21.2.7.16 **void LPUART\_TransferCreateHandle ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle*, lpuart\_transfer\_callback\_t *callback*, void \* *userData* )**

This function initializes the LPUART handle, which can be used for other LPUART transactional APIs. Usually, for a specified LPUART instance, call this API once to get the initialized handle.

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the [LPUART\\_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as *ringBuffer*.

## Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>callback</i>	Callback function.
<i>userData</i>	User data.

**21.2.7.17 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="#">Ipuart_transfer_t</a> .

## Return values

<i>kStatus_Success</i>	Successfully start the data transmission.
<i>kStatus_LPUART_TxBusy</i>	Previous transmission still not finished, data not all written to the TX register.
<i>kStatus_InvalidArgument</i>	Invalid argument.

**21.2.7.18 void LPUART\_TransferStartRingBuffer ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle*, uint8\_t \* *ringBuffer*, size\_t *ringBufferSize* )**

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the [UART\\_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

## LPUART Driver

### Note

When using RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, then only 31 bytes are used for saving data.

### Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>ringBuffer</i>	Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
<i>ringBufferSize</i>	size of the ring buffer.

### 21.2.7.19 void LPUART\_TransferStopRingBuffer ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle* )

This function aborts the background transfer and uninstalls the ring buffer.

### Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.

### 21.2.7.20 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 still not sent out.

### Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.

### 21.2.7.21 status\_t LPUART\_TransferGetSendCount ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been written to LPUART TX register by interrupt method.

## Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>count</i>	Send bytes count.

## Return values

<i>kStatus_NoTransferInProgress</i>	No send in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <i>count</i> ;

### 21.2.7.22 **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 #uart_transfer_t.
<i>receivedBytes</i>	Bytes received from the ring buffer directly.

## Return values

## LPUART Driver

<i>kStatus_Success</i>	Successfully queue the transfer into the transmit queue.
<i>kStatus_LPUART_Rx-Busy</i>	Previous receive request is not finished.
<i>kStatus_InvalidArgument</i>	Invalid argument.

### 21.2.7.23 void LPUART\_TransferAbortReceive ( **LPUART\_Type** \* *base*, **Ipuart\_handle\_t** \* *handle* )

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.

### 21.2.7.24 status\_t LPUART\_TransferGetReceiveCount ( **LPUART\_Type** \* *base*, **Ipuart\_handle\_t** \* *handle*, **uint32\_t** \* *count* )

This function gets the number of bytes that have been received.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>count</i>	Receive bytes count.

Return values

<i>kStatus_NoTransferIn-Progress</i>	No receive in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <i>count</i> ;

### 21.2.7.25 void LPUART\_TransferHandleIRQ ( **LPUART\_Type** \* *base*, **Ipuart\_handle\_t** \* *handle* )

This function handles the LPUART transmit and receive IRQ request.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.

#### 21.2.7.26 void LPUART\_TransferHandleErrorIRQ ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle* )

This function handles the LPUART error IRQ request.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.

### 21.3 LPUART DMA Driver

#### 21.3.1 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.*

#### 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)  
*Get the number of bytes that have been written to LPUART TX register.*
- status\_t [LPUART\\_TransferGetReceiveCountDMA](#) (LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been received.*

#### 21.3.2 Data Structure Documentation

##### 21.3.2.1 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.*

### 21.3.2.1.0.11 Field Documentation

21.3.2.1.0.11.1 `lpuart_dma_transfer_callback_t lpuart_dma_handle_t::callback`

21.3.2.1.0.11.2 `void* lpuart_dma_handle_t::userData`

21.3.2.1.0.11.3 `size_t lpuart_dma_handle_t::rxDataSizeAll`

21.3.2.1.0.11.4 `size_t lpuart_dma_handle_t::txDataSizeAll`

21.3.2.1.0.11.5 `dma_handle_t* lpuart_dma_handle_t::txDmaHandle`

21.3.2.1.0.11.6 `dma_handle_t* lpuart_dma_handle_t::rxDmaHandle`

21.3.2.1.0.11.7 `volatile uint8_t lpuart_dma_handle_t::txState`

### 21.3.3 Typedef Documentation

21.3.3.1 `typedef void(* lpuart_dma_transfer_callback_t)(LPUART_Type *base,  
lpuart_dma_handle_t *handle, status_t status, void *userData)`

### 21.3.4 Function Documentation

21.3.4.1 `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 )`

## LPUART DMA Driver

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	Pointer to lpuart_dma_handle_t 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.

### 21.3.4.2 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.

### 21.3.4.3 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.

#### 21.3.4.4 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

#### 21.3.4.5 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

#### 21.3.4.6 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.

## LPUART DMA Driver

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>count</i>	Send bytes count.

Return values

<i>kStatus_NoTransferInProgress</i>	No send in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <i>count</i> ;

### 21.3.4.7 **status\_t LPUART\_TransferGetReceiveCountDMA ( LPUART\_Type \* *base*, Ipuart\_dma\_handle\_t \* *handle*, uint32\_t \* *count* )**

This function gets the number of bytes that have been received.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>count</i>	Receive bytes count.

Return values

<i>kStatus_NoTransferInProgress</i>	No receive in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <i>count</i> ;

## 21.4 LPUART eDMA Driver

### 21.4.1 Overview

#### Data Structures

- struct [lpuart\\_edma\\_handle\\_t](#)  
*LPUART eDMA handle.* [More...](#)

#### TypeDefs

- [typedef void\(\\* lpuart\\_edma\\_transfer\\_callback\\_t \)](#)(LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, status\_t status, void \*userData)  
*LPUART transfer callback function.*

#### eDMA transactional

- void [LPUART\\_TransferCreateHandleEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, [lpuart\\_edma\\_transfer\\_callback\\_t](#) callback, void \*userData, [edma\\_handle\\_t](#) \*txEdmaHandle, [edma\\_handle\\_t](#) \*rxEdmaHandle)  
*Initializes the LPUART handle which is used in transactional functions.*
- status\_t [LPUART\\_SendEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, [lpuart\\_transfer\\_t](#) \*xfer)  
*Sends data using eDMA.*
- status\_t [LPUART\\_ReceiveEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, [lpuart\\_transfer\\_t](#) \*xfer)  
*Receives data using eDMA.*
- void [LPUART\\_TransferAbortSendEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle)  
*Aborts the sent data using eDMA.*
- void [LPUART\\_TransferAbortReceiveEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle)  
*Aborts the received data using eDMA.*
- status\_t [LPUART\\_TransferGetSendCountEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been written to LPUART TX register.*
- status\_t [LPUART\\_TransferGetReceiveCountEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been received.*

### 21.4.2 Data Structure Documentation

#### 21.4.2.1 struct \_lpuart\_edma\_handle

##### Data Fields

- `lpuart_edma_transfer_callback_t callback`  
*Callback function.*
- `void *userData`  
*LPUART callback function parameter.*
- `size_t rxDataSizeAll`  
*Size of the data to receive.*
- `size_t txDataSizeAll`  
*Size of the data to send out.*
- `edma_handle_t *txEdmaHandle`  
*The eDMA TX channel used.*
- `edma_handle_t *rxEdmaHandle`  
*The eDMA RX channel used.*
- `volatile uint8_t txState`  
*TX transfer state.*
- `volatile uint8_t rxState`  
*RX transfer state.*

#### 21.4.2.1.0.12 Field Documentation

21.4.2.1.0.12.1 `lpuart_edma_transfer_callback_t lpuart_edma_handle_t::callback`

21.4.2.1.0.12.2 `void* lpuart_edma_handle_t::userData`

21.4.2.1.0.12.3 `size_t lpuart_edma_handle_t::rxDataSizeAll`

21.4.2.1.0.12.4 `size_t lpuart_edma_handle_t::txDataSizeAll`

21.4.2.1.0.12.5 `edma_handle_t* lpuart_edma_handle_t::txEdmaHandle`

21.4.2.1.0.12.6 `edma_handle_t* lpuart_edma_handle_t::rxEdmaHandle`

21.4.2.1.0.12.7 `volatile uint8_t lpuart_edma_handle_t::txState`

#### 21.4.3 Typedef Documentation

21.4.3.1 `typedef void(* lpuart_edma_transfer_callback_t)(LPUART_Type *base,  
lpuart_edma_handle_t *handle, status_t status, void *userData)`

#### 21.4.4 Function Documentation

21.4.4.1 `void LPUART_TransferCreateHandleEDMA ( LPUART_Type * base,  
lpuart_edma_handle_t * handle, lpuart_edma_transfer_callback_t callback, void  
* userData, edma_handle_t * txEdmaHandle, edma_handle_t * rxEdmaHandle )`

## LPUART eDMA Driver

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	Pointer to lpuart_edma_handle_t structure.
<i>callback</i>	Callback function.
<i>userData</i>	User data.
<i>txEdmaHandle</i>	User requested DMA handle for TX DMA transfer.
<i>rxEdmaHandle</i>	User requested DMA handle for RX DMA transfer.

### 21.4.4.2 status\_t LPUART\_SendEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle*, lpuart\_transfer\_t \* *xfer* )

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>xfer</i>	LPUART eDMA transfer structure. See <a href="#">lpuart_transfer_t</a> .

Return values

<i>kStatus_Success</i>	if succeed, others failed.
<i>kStatus_LPUART_TxBusy</i>	Previous transfer on going.
<i>kStatus_InvalidArgument</i>	Invalid argument.

### 21.4.4.3 status\_t LPUART\_ReceiveEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle*, lpuart\_transfer\_t \* *xfer* )

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	Pointer to lpuart_edma_handle_t structure.
<i>xfer</i>	LPUART eDMA transfer structure, see <a href="#">lpuart_transfer_t</a> .

Return values

<i>kStatus_Success</i>	if succeed, others fail.
<i>kStatus_LPUART_Rx-Busy</i>	Previous transfer ongoing.
<i>kStatus_InvalidArgument</i>	Invalid argument.

#### 21.4.4.4 void LPUART\_TransferAbortSendEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle* )

This function aborts the sent data using eDMA.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	Pointer to lpuart_edma_handle_t structure.

#### 21.4.4.5 void LPUART\_TransferAbortReceiveEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle* )

This function aborts the received data using eDMA.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	Pointer to lpuart_edma_handle_t structure.

#### 21.4.4.6 status\_t LPUART\_TransferGetSendCountEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been written to LPUART TX register by DMA.

## LPUART eDMA Driver

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART 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> ;

### 21.4.4.7 **status\_t LPUART\_TransferGetReceiveCountEDMA ( LPUART\_Type \* *base*, Ipuart\_edma\_handle\_t \* *handle*, uint32\_t \* *count* )**

This function gets the number of bytes that have been received.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>count</i>	Receive bytes count.

Return values

<i>kStatus_NoTransferIn-Progress</i>	No receive in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <i>count</i> ;

## 21.5 LPUART µCOS/II Driver

### 21.5.1 Overview

#### LPUART RTOS Operation

- int **LPUART\_RTOS\_Init** (lpuart\_rtos\_handle\_t \*handle, lpuart\_handle\_t \*t\_handle, const struct rtos\_lpuart\_config \*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, const 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.*

### 21.5.2 Function Documentation

#### 21.5.2.1 int **LPUART\_RTOS\_Init** ( lpuart\_rtos\_handle\_t \* *handle*, lpuart\_handle\_t \* *t\_handle*, const struct rtos\_lpuart\_config \* *cfg* )

Parameters

<i>handle</i>	The RTOS LPUART handle, the pointer to an allocated space for RTOS context.
<i>lpuart_t_handle</i>	The pointer to an allocated space to store the transactional layer internal state.
<i>cfg</i>	The pointer to the parameters required to configure the LPUART after initialization.

Returns

0 succeed, others failed

#### 21.5.2.2 int **LPUART\_RTOS\_Deinit** ( lpuart\_rtos\_handle\_t \* *handle* )

This function deinitializes the LPUART module, sets all register values to the reset value, and releases the resources.

## LPUART µCOS/II Driver

Parameters

<i>handle</i>	The RTOS LPUART handle.
---------------	-------------------------

### **21.5.2.3 int LPUART\_RTOS\_Send ( Ipuart\_rtos\_handle\_t \* *handle*, const uint8\_t \* *buffer*, uint32\_t *length* )**

This function sends data. It is a synchronous API. If the hardware buffer is full, the task is in the blocked state.

Parameters

<i>handle</i>	The RTOS LPUART handle.
<i>buffer</i>	The pointer to buffer to send.
<i>length</i>	The number of bytes to send.

### **21.5.2.4 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 a synchronous API. If any data is immediately available it is returned immediately and the number of bytes received.

Parameters

<i>handle</i>	The RTOS LPUART handle.
<i>buffer</i>	The pointer to buffer where to write received data.
<i>length</i>	The number of bytes to receive.
<i>received</i>	The pointer to a variable of size_t where the number of received data is filled.

## 21.6 LPUART µCOS/III Driver

### 21.6.1 Overview

#### LPUART RTOS Operation

- int **LPUART\_RTOS\_Init** (lpuart\_rtos\_handle\_t \*handle, lpuart\_handle\_t \*t\_handle, const struct rtos\_lpuart\_config \*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, const uint8\_t \*buffer, uint32\_t length)  
*Send data in background.*
- int **LPUART\_RTOS\_Receive** (lpuart\_rtos\_handle\_t \*handle, uint8\_t \*buffer, uint32\_t length, size\_t \*received)  
*Receives data.*

### 21.6.2 Function Documentation

#### 21.6.2.1 int **LPUART\_RTOS\_Init** ( lpuart\_rtos\_handle\_t \* *handle*, lpuart\_handle\_t \* *t\_handle*, const struct rtos\_lpuart\_config \* *cfg* )

Parameters

<i>handle</i>	The RTOS LPUART handle, the pointer to allocated space for RTOS context.
<i>lpuart_t_handle</i>	The pointer to allocated space where to store transactional layer internal state.
<i>cfg</i>	The pointer to the parameters required to configure the LPUART after initialization.

Returns

0 succeed, others failed

#### 21.6.2.2 int **LPUART\_RTOS\_Deinit** ( lpuart\_rtos\_handle\_t \* *handle* )

This function deinitializes the LPUART module, set all register value to reset value and releases the resources.

## LPUART µCOS/III Driver

Parameters

<i>handle</i>	The RTOS LPUART handle.
---------------	-------------------------

### **21.6.2.3 int LPUART\_RTOS\_Send ( *Ipuart\_rtos\_handle\_t* \* *handle*, *const uint8\_t* \* *buffer*, *uint32\_t* *length* )**

This function sends data. It is synchronous API. If the HW buffer is full, the task is in the blocked state.

Parameters

<i>handle</i>	The RTOS LPUART handle.
<i>buffer</i>	The pointer to buffer to send.
<i>length</i>	The number of bytes to send.

### **21.6.2.4 int LPUART\_RTOS\_Receive ( *Ipuart\_rtos\_handle\_t* \* *handle*, *uint8\_t* \* *buffer*, *uint32\_t* *length*, *size\_t* \* *received* )**

It is synchronous API.

This function receives data from LPUART. If any data is immediately available it will be returned imidiately 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 variable of <i>size_t</i> where the number of received data will be filled.

## 21.7 LPUART FreeRTOS Driver

### 21.7.1 Overview

#### LPUART RTOS Operation

- int **LPUART\_RTOs\_Init** (lpuart\_rtos\_handle\_t \*handle, lpuart\_handle\_t \*t\_handle, const struct rtos\_lpuart\_config \*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, const uint8\_t \*buffer, uint32\_t length)  
*Sends data in background.*
- int **LPUART\_RTOs\_Receive** (lpuart\_rtos\_handle\_t \*handle, uint8\_t \*buffer, uint32\_t length, size\_t \*received)  
*Receives data.*

### 21.7.2 Function Documentation

#### 21.7.2.1 int **LPUART\_RTOs\_Init** ( lpuart\_rtos\_handle\_t \* *handle*, lpuart\_handle\_t \* *t\_handle*, const struct rtos\_lpuart\_config \* *cfg* )

Parameters

<i>handle</i>	The RTOS LPUART handle, the pointer to an allocated space for RTOS context.
<i>t_handle</i>	The pointer to an allocated space to store the transactional layer internal state.
<i>cfg</i>	The pointer to the parameters required to configure the LPUART after initialization.

Returns

0 succeed, others failed

#### 21.7.2.2 int **LPUART\_RTOs\_Deinit** ( lpuart\_rtos\_handle\_t \* *handle* )

This function deinitializes the LPUART module, sets all register value to the reset value, and releases the resources.

## LPUART FreeRTOS Driver

Parameters

<i>handle</i>	The RTOS LPUART handle.
---------------	-------------------------

### 21.7.2.3 int LPUART\_RTOS\_Send ( *Ipuart\_rtos\_handle\_t \* handle*, *const uint8\_t \* buffer*, *uint32\_t length* )

This function sends data. It is an synchronous API. If the hardware buffer is full, the task is in the blocked state.

Parameters

<i>handle</i>	The RTOS LPUART handle.
<i>buffer</i>	The pointer to buffer to send.
<i>length</i>	The number of bytes to send.

### 21.7.2.4 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.

# Chapter 22

## LTC: LP Trusted Cryptography

### 22.1 Overview

The Kinetis SDK provides the Peripheral driver for the LP Trusted Cryptography (LTC) module of Kinetis devices. LP Trusted Cryptography is a set of cryptographpic hardware accelerator engines that share common registers. LTC architecture can support AES, DES, 3DES, MDHA (SHA), RSA and ECC. Actual list of implemented cryptographpic hardware accelerator engines depends on specific Kinetis microcontroller.

The driver comprises two sets of API functions.

In the first set, blocking synchronous APIs are provided, for all operations supported by LTC hardware. The LTC operations are complete (and results are made available for further usage) when a function returns. When called, these functions don't return until an LTC operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status and also for plaintext or ciphertext data movements. The driver functions are not re-entrant. These functions provide typical interface to upper layer or application software.

In the second set, DMA support for symmetric LTC processing is provided, for AES and DES engines. APIs in the second set use DMA for data movement to and from the LTC input and output FIFOs. By using these functions, main CPU is not used for plaintext or ciphertext data movements (DMA is used instead). Thus, CPU processing power can be used for other application tasks, at cost of decreased maximum data throughput (because of DMA module and transactions management overhead). These functions provide less typical interface, for applications that must offload main CPU while ciphertext or plaintext is being processed, at cost of longer cryptographpic processing time.

### 22.2 LTC Driver Initialization and Configuration

LTC Driver is initialized by calling the [LTC\\_Init\(\)](#) function, it enables the LTC module clock in the Kinetis SIM module. If AES or DES engine is used and the LTC module implementation features the LTC DPA Mask Seed register, seed the DPA mask generator by using the seed from a random number generator. The [LTC\\_SetDpaMaskSeed\(\)](#) function is provided to set the DPA mask seed.

### 22.3 Comments about API usage in RTOS

LTC operations provided by this driver are not re-entrant. Thus, application software shall ensure the LTC module operation is not requested from different tasks or interrupt service routines while an operation is in progress.

### 22.4 Comments about API usage in interrupt handler

All APIs can be used from interrupt handler although execution time shall be considered (interrupt latency of equal and lower priority interrupts increases).

## LTC Driver Examples

### 22.5 LTC Driver Examples

#### 22.5.1 Simple examples

Initialize LTC after Power On Reset or reset cycle

```
LTC_Init(LTC0);
/* optionally initialize DPA mask seed register */
LTC_SetDpaMaskSeed(randomNumber);
```

Encrypt plaintext by DES engine

```
char plain[16];
char cipher[16];

char iv[LTC_DES_IV_SIZE];
char key1[LTC_DES_KEY_SIZE];
char key2[LTC_DES_KEY_SIZE];
char key3[LTC_DES_KEY_SIZE];

memcpy(plain, "Hello World!", 12);
memcpy(iv, "initvect", LTC_DES_IV_SIZE);
memcpy(key1, "mykey1aa", LTC_DES_KEY_SIZE);
memcpy(key2, "mykey2bb", LTC_DES_KEY_SIZE);
memcpy(key3, "mykey3cc", LTC_DES_KEY_SIZE);

LTC_DES3_EncryptCbc(LTC0, plain, cipher, 16, iv, key1, key2, key3);
```

Encrypt plaintext by AES engine

```
char plain[16] = {0};
char cipher[16];
char iv[16] = {0};
char key[16] = {0};

memcpy(plain, "Hello World!", 12);
memcpy(iv, "initvectorinitve", 16);
memcpy(key, "__mykey1aa__^^..", 16);

LTC_AES_EncryptCbc(LTC0, plain, cipher, 16, iv, key, 16);
```

Compute keyed hash by AES engine (CMAC)

```
char message[] = "Hello World!";
char key[16] = {0};
char output[16];
uint32_t szOutput = 16u;

memcpy(key, "__mykey1aa__^^..", 16);
LTC_HASH(LTC0, kLTC_Sha256, message, sizeof(message), NULL, 0, output, &szOutput);
```

Compute hash by MDHA engine (SHA-256)

```
char message[] = "Hello World!";
char output[32];
uint32_t szOutput = 32u;

LTC_HASH(LTC0, kLTC_Sha256, message, sizeof(message), NULL, output, &szOutput);
```

## Compute modular integer exponentiation

```

status_t status;
const char bigA[] = "112233445566778899aabcccddeeff";
const char bigN[] = "aabbaabbabb112233445566778899aabcccddeefe";
const char bigE[] = "065537";
char A[256], E[256], N[256], res[256];
uint16_t sizeA, sizeE, sizeN, sizeRes;

/* Note LTC PKHA integer format is least significant byte at lowest address.
 * The _read_string() function converts the input string to LTC PKHA integer format
 * and writes sizeof() the integer to the size variable (sizeA, sizeE, sizeN).
 */
_read_string(A, &sizeA, bigA);
_read_string(E, &sizeE, bigN);
_read_string(N, &sizeN, bigE);

status = LTC_PKHA_ModExp(base, A, sizeA, N, sizeN, E, sizeE, res, &sizeRes,
    kLTC_PKHA_IntegerArith,
    kLTC_PKHA_NormalValue,
    kLTC_PKHA_TimingEqualized);

```

## Compute elliptic curve point multiplication

```

status_t status;
ltc_pkha_ecc_point_t B0, res0;
uint8_t bx, by, resx, resy;
uint8_t E[256];
bool isPointOfInfinity;
uint16_t resultSize, sizeE;

/* Example carried out with 1-byte curve params and point coordinates. */
uint8_t size = 1;
uint8_t aCurveParam = 1;
uint8_t bCurveParam = 0;

bx = 9;
by = 5;

B0.X = &bx;
B0.Y = &by;
res0.X = &resx;
res0.Y = &resy;

/* Prime modulus of the field. */
N[0] = 23;

/* Note LTC PKHA integer has least significant byte at lowest address */

/* Scalar multiplier */
char ew[] = "0100"; /* 256 in decimal */
_read_string(E, &sizeE, ew);

status = LTC_PKHA_ECC_PointMul(LTC0, &B0, E, sizeE, N, NULL, &aCurveParam, &
    bCurveParam, size,
    kLTC_PKHA_TimingEqualized,
    kLTC_PKHA_IntegerArith, &res0, &isPointOfInfinity);

```

## Modules

- [LTC Blocking APIs](#)
- [LTC Non-blocking eDMA APIs](#)

## Function Documentation

### Functions

- void **LTC\_Init** (LTC\_Type \*base)  
*Initializes the LTC driver.*
- void **LTC\_Deinit** (LTC\_Type \*base)  
*Deinitializes the LTC driver.*
- void **LTC\_SetDpaMaskSeed** (LTC\_Type \*base, uint32\_t mask)  
*Sets the DPA Mask Seed register.*

### Driver version

- #define **FSL\_LTC\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 1))  
*LTC driver version.*

## 22.6 Macro Definition Documentation

### 22.6.1 #define FSL\_LTC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

Version 2.0.1.

Current version: 2.0.1

Change log:

- Version 2.0.1
  - fixed warning during g++ compilation

## 22.7 Function Documentation

### 22.7.1 void LTC\_Init ( LTC\_Type \* *base* )

This function initializes the LTC driver.

Parameters

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

### 22.7.2 void LTC\_Deinit ( LTC\_Type \* *base* )

This function deinitializes the LTC driver.

Parameters

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

### 22.7.3 void LTC\_SetDpaMaskSeed ( LTC\_Type \* *base*, uint32\_t *mask* )

The DPA Mask Seed register reseeds the mask that provides resistance against DPA (differential power analysis) attacks on AES or DES keys.

Differential Power Analysis Mask (DPA) resistance uses a randomly changing mask that introduces "noise" into the power consumed by the AES or DES. This reduces the signal-to-noise ratio that differential power analysis attacks use to "guess" bits of the key. This randomly changing mask should be seeded at POR, and continues to provide DPA resistance from that point on. However, to provide even more DPA protection it is recommended that the DPA mask be reseeded after every 50,000 blocks have been processed. At that time, software can opt to write a new seed (preferably obtained from an RNG) into the DPA Mask Seed register (DPAMS), or software can opt to provide the new seed earlier or later, or not at all. DPA resistance continues even if the DPA mask is never reseeded.

Parameters

<i>base</i>	LTC peripheral base address
<i>mask</i>	The DPA mask seed.

## LTC Blocking APIs

### 22.8 LTC Blocking APIs

#### 22.8.1 Overview

This section describes the programming interface of the LTC Synchronous Blocking functions

#### Modules

- [LTC AES driver](#)
- [LTC DES driver](#)
- [LTC HASH driver](#)
- [LTC PKHA driver](#)

## 22.8.2 LTC DES driver

### 22.8.2.1 Overview

This section describes the programming interface of the LTC DES driver.

#### Macros

- #define `LTC_DES_KEY_SIZE` 8  
*LTC DES key size - 64 bits.*
- #define `LTC_DES_IV_SIZE` 8  
*LTC DES IV size - 8 bytes.*

#### Functions

- status\_t `LTC_DES_EncryptEcb` (LTC\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t key[`LTC_DES_KEY_SIZE`])  
*Encrypts DES using ECB block mode.*
- status\_t `LTC_DES_DecryptEcb` (LTC\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t key[`LTC_DES_KEY_SIZE`])  
*Decrypts DES using ECB block mode.*
- status\_t `LTC_DES_EncryptCbc` (LTC\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t iv[`LTC_DES_IV_SIZE`], const uint8\_t key[`LTC_DES_KEY_SIZE`])  
*Encrypts DES using CBC block mode.*
- status\_t `LTC_DES_DecryptCbc` (LTC\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[`LTC_DES_IV_SIZE`], const uint8\_t key[`LTC_DES_KEY_SIZE`])  
*Decrypts DES using CBC block mode.*
- status\_t `LTC_DES_EncryptCfb` (LTC\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t iv[`LTC_DES_IV_SIZE`], const uint8\_t key[`LTC_DES_KEY_SIZE`])  
*Encrypts DES using CFB block mode.*
- status\_t `LTC_DES_DecryptCfb` (LTC\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[`LTC_DES_IV_SIZE`], const uint8\_t key[`LTC_DES_KEY_SIZE`])  
*Decrypts DES using CFB block mode.*
- status\_t `LTC_DES_EncryptOfb` (LTC\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t iv[`LTC_DES_IV_SIZE`], const uint8\_t key[`LTC_DES_KEY_SIZE`])  
*Encrypts DES using OFB block mode.*
- status\_t `LTC_DES_DecryptOfb` (LTC\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[`LTC_DES_IV_SIZE`], const uint8\_t key[`LTC_DES_KEY_SIZE`])  
*Decrypts DES using OFB block mode.*
- status\_t `LTC_DES2_EncryptEcb` (LTC\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t key1[`LTC_DES_KEY_SIZE`], const uint8\_t key2[`LTC_DES_KEY_SIZE`])  
*Encrypts triple DES using ECB block mode with two keys.*
- status\_t `LTC_DES2_DecryptEcb` (LTC\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t key1[`LTC_DES_KEY_SIZE`], const uint8\_t key2[`LTC_DES_KEY_SIZE`])  
*Decrypts triple DES using ECB block mode with two keys.*

## LTC Blocking APIs

- status\_t [LTC\\_DES2\\_EncryptCbc](#) (LTC\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE])  
*Encrypts triple DES using CBC block mode with two keys.*
- status\_t [LTC\\_DES2\\_DecryptCbc](#) (LTC\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE])  
*Decrypts triple DES using CBC block mode with two keys.*
- status\_t [LTC\\_DES2\\_EncryptCfb](#) (LTC\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE])  
*Encrypts triple DES using CFB block mode with two keys.*
- status\_t [LTC\\_DES2\\_DecryptCfb](#) (LTC\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE])  
*Decrypts triple DES using CFB block mode with two keys.*
- status\_t [LTC\\_DES2\\_EncryptOfb](#) (LTC\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE])  
*Encrypts triple DES using OFB block mode with two keys.*
- status\_t [LTC\\_DES2\\_DecryptOfb](#) (LTC\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE])  
*Decrypts triple DES using OFB block mode with two keys.*
- status\_t [LTC\\_DES3\\_EncryptEcb](#) (LTC\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE], const uint8\_t key3[LTC\_DES\_KEY\_SIZE])  
*Encrypts triple DES using ECB block mode with three keys.*
- status\_t [LTC\\_DES3\\_DecryptEcb](#) (LTC\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE], const uint8\_t key3[LTC\_DES\_KEY\_SIZE])  
*Decrypts triple DES using ECB block mode with three keys.*
- status\_t [LTC\\_DES3\\_EncryptCbc](#) (LTC\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE], const uint8\_t key3[LTC\_DES\_KEY\_SIZE])  
*Encrypts triple DES using CBC block mode with three keys.*
- status\_t [LTC\\_DES3\\_DecryptCbc](#) (LTC\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE], const uint8\_t key3[LTC\_DES\_KEY\_SIZE])  
*Decrypts triple DES using CBC block mode with three keys.*
- status\_t [LTC\\_DES3\\_EncryptCfb](#) (LTC\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE], const uint8\_t key3[LTC\_DES\_KEY\_SIZE])  
*Encrypts triple DES using CFB block mode with three keys.*
- status\_t [LTC\\_DES3\\_DecryptCfb](#) (LTC\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE], const uint8\_t key3[LTC\_DES\_KEY\_SIZE])  
*Decrypts triple DES using CFB block mode with three keys.*

- status\_t **LTC\_DES3\_EncryptOfb** (LTC\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE], const uint8\_t key3[LTC\_DES\_KEY\_SIZE])  
*Encrypts triple DES using OFB block mode with three keys.*
- status\_t **LTC\_DES3\_DecryptOfb** (LTC\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE], const uint8\_t key3[LTC\_DES\_KEY\_SIZE])  
*Decrypts triple DES using OFB block mode with three keys.*

## 22.8.2.2 Macro Definition Documentation

### 22.8.2.2.1 #define LTC\_DES\_KEY\_SIZE 8

## 22.8.2.3 Function Documentation

### 22.8.2.3.1 status\_t LTC\_DES\_EncryptEcb ( LTC\_Type \* *base*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *key*[LTC\_DES\_KEY\_SIZE] )

Encrypts DES using ECB block mode.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key</i>	Input key to use for encryption

Returns

Status from encrypt/decrypt operation

### 22.8.2.3.2 status\_t LTC\_DES\_DecryptEcb ( LTC\_Type \* *base*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t *key*[LTC\_DES\_KEY\_SIZE] )

Decrypts DES using ECB block mode.

Parameters

## LTC Blocking APIs

	<i>base</i>	LTC peripheral base address
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key</i>	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

**22.8.2.3.3 status\_t LTC\_DES\_EncryptCbc ( LTC\_Type \* *base*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key*[LTC\_DES\_KEY\_SIZE] )**

Encrypts DES using CBC block mode.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Ouput ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key</i>	Input key to use for encryption

Returns

Status from encrypt/decrypt operation

**22.8.2.3.4 status\_t LTC\_DES\_DecryptCbc ( LTC\_Type \* *base*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key*[LTC\_DES\_KEY\_SIZE] )**

Decrypts DES using CBC block mode.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key</i>	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

**22.8.2.3.5 status\_t LTC\_DES\_EncryptCfb ( LTC\_Type \* *base*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key*[LTC\_DES\_KEY\_SIZE] )**

Encrypts DES using CFB block mode.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>plaintext</i>	Input plaintext to encrypt
	<i>size</i>	Size of input data in bytes
	<i>iv</i>	Input initial block.
	<i>key</i>	Input key to use for encryption
out	<i>ciphertext</i>	Output ciphertext

Returns

Status from encrypt/decrypt operation

**22.8.2.3.6 status\_t LTC\_DES\_DecryptCfb ( LTC\_Type \* *base*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key*[LTC\_DES\_KEY\_SIZE] )**

Decrypts DES using CFB block mode.

## LTC Blocking APIs

Parameters

	<i>base</i>	LTC peripheral base address
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial block.
	<i>key</i>	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

**22.8.2.3.7 status\_t LTC\_DES\_EncryptOfb ( LTC\_Type \* *base*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key*[LTC\_DES\_KEY\_SIZE] )**

Encrypts DES using OFB block mode.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key</i>	Input key to use for encryption

Returns

Status from encrypt/decrypt operation

**22.8.2.3.8 status\_t LTC\_DES\_DecryptOfb ( LTC\_Type \* *base*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key*[LTC\_DES\_KEY\_SIZE] )**

Decrypts DES using OFB block mode.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key</i>	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

**22.8.2.3.9 status\_t LTC\_DES2\_EncryptEcb ( LTC\_Type \* *base*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE] )**

Encrypts triple DES using ECB block mode with two keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

**22.8.2.3.10 status\_t LTC\_DES2\_DecryptEcb ( LTC\_Type \* *base*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE] )**

Decrypts triple DES using ECB block mode with two keys.

## LTC Blocking APIs

Parameters

	<i>base</i>	LTC peripheral base address
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

**22.8.2.3.11 status\_t LTC\_DES2\_EncryptCbc ( LTC\_Type \* *base*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE] )**

Encrypts triple DES using CBC block mode with two keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

**22.8.2.3.12 status\_t LTC\_DES2\_DecryptCbc ( LTC\_Type \* *base*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE] )**

Decrypts triple DES using CBC block mode with two keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

**22.8.2.3.13 status\_t LTC\_DES2\_EncryptCfb ( LTC\_Type \* *base*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE] )**

Encrypts triple DES using CFB block mode with two keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial block.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

**22.8.2.3.14 status\_t LTC\_DES2\_DecryptCfb ( LTC\_Type \* *base*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE] )**

Decrypts triple DES using CFB block mode with two keys.

## LTC Blocking APIs

Parameters

	<i>base</i>	LTC peripheral base address
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial block.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

**22.8.2.3.15 status\_t LTC\_DES2\_EncryptOfb ( LTC\_Type \* *base*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE] )**

Encrypts triple DES using OFB block mode with two keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

**22.8.2.3.16 status\_t LTC\_DES2\_DecryptOfb ( LTC\_Type \* *base*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE] )**

Decrypts triple DES using OFB block mode with two keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

**22.8.2.3.17 status\_t LTC\_DES3\_EncryptEcb ( LTC\_Type \* *base*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key3*[LTC\_DES\_KEY\_SIZE] )**

Encrypts triple DES using ECB block mode with three keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

**22.8.2.3.18 status\_t LTC\_DES3\_DecryptEcb ( LTC\_Type \* *base*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key3*[LTC\_DES\_KEY\_SIZE] )**

Decrypts triple DES using ECB block mode with three keys.

## LTC Blocking APIs

Parameters

	<i>base</i>	LTC peripheral base address
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

**22.8.2.3.19 status\_t LTC\_DES3\_EncryptCbc ( LTC\_Type \* *base*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key3*[LTC\_DES\_KEY\_SIZE] )**

Encrypts triple DES using CBC block mode with three keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
22.8.2.3.20 status_t LTC_DES3_DecryptCbc ( LTC_Type * base, const uint8_t * ciphertext,  
    uint8_t * plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t  
    key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t  
    key3[LTC_DES_KEY_SIZE] )
```

Decrypts triple DES using CBC block mode with three keys.

## LTC Blocking APIs

Parameters

	<i>base</i>	LTC peripheral base address
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
22.8.2.3.21 status_t LTC_DES3_EncryptCfb ( LTC_Type * base, const uint8_t * plaintext, uint8_t  
* ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t  
key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t  
key3[LTC_DES_KEY_SIZE] )
```

Encrypts triple DES using CFB block mode with three keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and ouput data in bytes
	<i>iv</i>	Input initial block.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
22.8.2.3.22 status_t LTC_DES3_DecryptCfb ( LTC_Type * base, const uint8_t * ciphertext,  
    uint8_t * plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t  
    key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t  
    key3[LTC_DES_KEY_SIZE] )
```

Decrypts triple DES using CFB block mode with three keys.

## LTC Blocking APIs

Parameters

	<i>base</i>	LTC peripheral base address
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input data in bytes
	<i>iv</i>	Input initial block.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

**22.8.2.3.23 status\_t LTC\_DES3\_EncryptOfb ( LTC\_Type \* *base*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key3*[LTC\_DES\_KEY\_SIZE] )**

Encrypts triple DES using OFB block mode with three keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
22.8.2.3.24 status_t LTC_DES3_DecryptOfb ( LTC_Type * base, const uint8_t * ciphertext,  
    uint8_t * plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t  
    key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t  
    key3[LTC_DES_KEY_SIZE] )
```

Decrypts triple DES using OFB block mode with three keys.

## LTC Blocking APIs

### Parameters

	<i>base</i>	LTC peripheral base address
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

### Returns

Status from encrypt/decrypt operation

## 22.8.3 LTC AES driver

### 22.8.3.1 Overview

This section describes the programming interface of the LTC AES driver.

#### Macros

- `#define LTC_AES_BLOCK_SIZE 16`  
*AES block size in bytes.*
- `#define LTC_AES_IV_SIZE 16`  
*AES Input Vector size in bytes.*
- `#define LTC_AES_DecryptCtr(base, input, output, size, counter, key, keySize, counterlast, szLeft) LTC_AES_CryptCtr(base, input, output, size, counter, key, keySize, counterlast, szLeft)`  
*AES CTR decrypt is mapped to the AES CTR generic operation.*
- `#define LTC_AES_EncryptCtr(base, input, output, size, counter, key, keySize, counterlast, szLeft) LTC_AES_CryptCtr(base, input, output, size, counter, key, keySize, counterlast, szLeft)`  
*AES CTR encrypt is mapped to the AES CTR generic operation.*

#### Enumerations

- `enum ltc_aes_key_t {`  
  `kLTC_EncryptKey = 0U,`  
  `kLTC_DecryptKey = 1U }`  
*Type of AES key for ECB and CBC decrypt operations.*

#### Functions

- `status_t LTC_AES_GenerateDecryptKey (LTC_Type *base, const uint8_t *encryptKey, uint8_t *decryptKey, uint32_t keySize)`  
*Transforms an AES encrypt key (forward AES) into the decrypt key (inverse AES).*
- `status_t LTC_AES_EncryptEcb (LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t *key, uint32_t keySize)`  
*Encrypts AES using the ECB block mode.*
- `status_t LTC_AES_DecryptEcb (LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t *key, uint32_t keySize, ltc_aes_key_t keyType)`  
*Decrypts AES using ECB block mode.*
- `status_t LTC_AES_EncryptCbc (LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_AES_IV_SIZE], const uint8_t *key, uint32_t keySize)`  
*Encrypts AES using CBC block mode.*
- `status_t LTC_AES_DecryptCbc (LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_AES_IV_SIZE], const uint8_t *key, uint32_t keySize, ltc_aes_key_t keyType)`  
*Decrypts AES using CBC block mode.*
- `status_t LTC_AES_CryptCtr (LTC_Type *base, const uint8_t *input, uint8_t *output, uint32_t size, uint8_t counter[LTC_AES_BLOCK_SIZE], const uint8_t *key, uint32_t keySize, uint8_t`

## LTC Blocking APIs

counterlast[LTC\_AES\_BLOCK\_SIZE], uint32\_t \*szLeft)

*Encrypts or decrypts AES using CTR block mode.*

- status\_t LTC\_AES\_EncryptTagGcm (LTC\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t \*iv, uint32\_t ivSize, const uint8\_t \*aad, uint32\_t aadSize, const uint8\_t \*key, uint32\_t keySize, uint8\_t \*tag, uint32\_t tagSize)

*Encrypts AES and tags using GCM block mode.*

- status\_t LTC\_AES\_DecryptTagGcm (LTC\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t \*iv, uint32\_t ivSize, const uint8\_t \*aad, uint32\_t aadSize, const uint8\_t \*key, uint32\_t keySize, const uint8\_t \*tag, uint32\_t tagSize)

*Decrypts AES and authenticates using GCM block mode.*

- status\_t LTC\_AES\_EncryptTagCcm (LTC\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t \*iv, uint32\_t ivSize, const uint8\_t \*aad, uint32\_t aadSize, const uint8\_t \*key, uint32\_t keySize, uint8\_t \*tag, uint32\_t tagSize)

*Encrypts AES and tags using CCM block mode.*

- status\_t LTC\_AES\_DecryptTagCcm (LTC\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t \*iv, uint32\_t ivSize, const uint8\_t \*aad, uint32\_t aadSize, const uint8\_t \*key, uint32\_t keySize, const uint8\_t \*tag, uint32\_t tagSize)

*Decrypts AES and authenticates using CCM block mode.*

### 22.8.3.2 Enumeration Type Documentation

#### 22.8.3.2.1 enum ltc\_aes\_key\_t

Enumerator

*kLTC\_EncryptKey* Input key is an encrypt key.

*kLTC\_DecryptKey* Input key is a decrypt key.

### 22.8.3.3 Function Documentation

#### 22.8.3.3.1 status\_t LTC\_AES\_GenerateDecryptKey ( LTC\_Type \* *base*, const uint8\_t \* *encryptKey*, uint8\_t \* *decryptKey*, uint32\_t *keySize* )

Transforms the AES encrypt key (forward AES) into the decrypt key (inverse AES). The key derived by this function can be used as a direct load decrypt key for AES ECB and CBC decryption operations (keyType argument).

Parameters

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

	<i>encryptKey</i>	Input key for decrypt key transformation
out	<i>decryptKey</i>	Output key, the decrypt form of the AES key.
	<i>keySize</i>	Size of the input key and output key in bytes. Must be 16, 24, or 32.

Returns

Status from key generation operation

#### 22.8.3.3.2 **status\_t LTC\_AES\_EncryptEcb ( LTC\_Type \* *base*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t \* *key*, uint32\_t *keySize* )**

Encrypts AES using the ECB block mode.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>plaintext</i>	Input plain text to encrypt
out	<i>ciphertext</i>	Output cipher text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>key</i>	Input key to use for encryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from encrypt operation

#### 22.8.3.3.3 **status\_t LTC\_AES\_DecryptEcb ( LTC\_Type \* *base*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t \* *key*, uint32\_t *keySize*, ltc\_aes\_key\_t *keyType* )**

Decrypts AES using ECB block mode.

Parameters

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

## LTC Blocking APIs

	<i>ciphertext</i>	Input cipher text to decrypt
out	<i>plaintext</i>	Output plain text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>key</i>	Input key.
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
	<i>keyType</i>	Input type of the key (allows to directly load decrypt key for AES ECB decrypt operation.)

Returns

Status from decrypt operation

**22.8.3.3.4 status\_t LTC\_AES\_EncryptCbc ( LTC\_Type \* *base*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_AES\_IV\_SIZE], const uint8\_t \* *key*, uint32\_t *keySize* )**

Parameters

	<i>base</i>	LTC peripheral base address
	<i>plaintext</i>	Input plain text to encrypt
out	<i>ciphertext</i>	Output cipher text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>iv</i>	Input initial vector to combine with the first input block.
	<i>key</i>	Input key to use for encryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from encrypt operation

**22.8.3.3.5 status\_t LTC\_AES\_DecryptCbc ( LTC\_Type \* *base*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_AES\_IV\_SIZE], const uint8\_t \* *key*, uint32\_t *keySize*, ltc\_aes\_key\_t *keyType* )**

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>ciphertext</i>	Input cipher text to decrypt
out	<i>plaintext</i>	Output plain text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>iv</i>	Input initial vector to combine with the first input block.
	<i>key</i>	Input key to use for decryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
	<i>keyType</i>	Input type of the key (allows to directly load decrypt key for AES CBC decrypt operation.)

## Returns

Status from decrypt operation

**22.8.3.3.6 status\_t LTC\_AES\_CryptCtr ( LTC\_Type \* *base*, const uint8\_t \* *input*, uint8\_t \* *output*, uint32\_t *size*, uint8\_t *counter*[LTC\_AES\_BLOCK\_SIZE], const uint8\_t \* *key*, uint32\_t *keySize*, uint8\_t *counterlast*[LTC\_AES\_BLOCK\_SIZE], uint32\_t \* *szLeft* )**

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>input</i>	Input data for CTR block mode
out	<i>output</i>	Output data for CTR block mode
	<i>size</i>	Size of input and output data in bytes
in, out	<i>counter</i>	Input counter (updates on return)
	<i>key</i>	Input key to use for forward AES cipher

## LTC Blocking APIs

	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
out	<i>counterlast</i>	Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
out	<i>szLeft</i>	Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.

Returns

Status from encrypt operation

**22.8.3.3.7 status\_t LTC\_AES\_EncryptTagGcm ( LTC\_Type \* *base*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t \* *iv*, uint32\_t *ivSize*, const uint8\_t \* *aad*, uint32\_t *aadSize*, const uint8\_t \* *key*, uint32\_t *keySize*, uint8\_t \* *tag*, uint32\_t *tagSize* )**

Encrypts AES and optionally tags using GCM block mode. If plaintext is NULL, only the GHASH is calculated and output in the 'tag' field.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>plaintext</i>	Input plain text to encrypt
out	<i>ciphertext</i>	Output cipher text.
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector
	<i>ivSize</i>	Size of the IV
	<i>aad</i>	Input additional authentication data
	<i>aadSize</i>	Input size in bytes of AAD
	<i>key</i>	Input key to use for encryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
out	<i>tag</i>	Output hash tag. Set to NULL to skip tag processing.
	<i>tagSize</i>	Input size of the tag to generate, in bytes. Must be 4,8,12,13,14,15 or 16.

Returns

Status from encrypt operation

```
22.8.3.3.8 status_t LTC_AES_DecryptTagGcm ( LTC_Type * base, const uint8_t * ciphertext,  
    uint8_t * plaintext, uint32_t size, const uint8_t * iv, uint32_t ivSize, const uint8_t *  
    aad, uint32_t aadSize, const uint8_t * key, uint32_t keySize, const uint8_t * tag,  
    uint32_t tagSize )
```

Decrypts AES and optionally authenticates using GCM block mode. If ciphertext is NULL, only the GHASH is calculated and compared with the received GHASH in 'tag' field.

## LTC Blocking APIs

Parameters

	<i>base</i>	LTC peripheral base address
	<i>ciphertext</i>	Input cipher text to decrypt
out	<i>plaintext</i>	Output plain text.
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector
	<i>ivSize</i>	Size of the IV
	<i>aad</i>	Input additional authentication data
	<i>aadSize</i>	Input size in bytes of AAD
	<i>key</i>	Input key to use for encryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
	<i>tag</i>	Input hash tag to compare. Set to NULL to skip tag processing.
	<i>tagSize</i>	Input size of the tag, in bytes. Must be 4, 8, 12, 13, 14, 15, or 16.

Returns

Status from decrypt operation

**22.8.3.3.9 status\_t LTC\_AES\_EncryptTagCcm ( LTC\_Type \* *base*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t \* *iv*, uint32\_t *ivSize*, const uint8\_t \* *aad*, uint32\_t *aadSize*, const uint8\_t \* *key*, uint32\_t *keySize*, uint8\_t \* *tag*, uint32\_t *tagSize* )**

Encrypts AES and optionally tags using CCM block mode.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>plaintext</i>	Input plain text to encrypt
out	<i>ciphertext</i>	Output cipher text.
	<i>size</i>	Size of input and output data in bytes. Zero means authentication only.
	<i>iv</i>	Nonce

	<i>ivSize</i>	Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
	<i>aad</i>	Input additional authentication data. Can be NULL if aadSize is zero.
	<i>aadSize</i>	Input size in bytes of AAD. Zero means data mode only (authentication skipped).
	<i>key</i>	Input key to use for encryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
out	<i>tag</i>	Generated output tag. Set to NULL to skip tag processing.
	<i>tagSize</i>	Input size of the tag to generate, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.

Returns

Status from encrypt operation

**22.8.3.3.10 status\_t LTC\_AES\_DecryptTagCcm ( LTC\_Type \* *base*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t \* *iv*, uint32\_t *ivSize*, const uint8\_t \* *aad*, uint32\_t *aadSize*, const uint8\_t \* *key*, uint32\_t *keySize*, const uint8\_t \* *tag*, uint32\_t *tagSize* )**

Decrypts AES and optionally authenticates using CCM block mode.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>ciphertext</i>	Input cipher text to decrypt
out	<i>plaintext</i>	Output plain text.
	<i>size</i>	Size of input and output data in bytes. Zero means authentication only.
	<i>iv</i>	Nonce
	<i>ivSize</i>	Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
	<i>aad</i>	Input additional authentication data. Can be NULL if aadSize is zero.
	<i>aadSize</i>	Input size in bytes of AAD. Zero means data mode only (authentication skipped).

## LTC Blocking APIs

	<i>key</i>	Input key to use for decryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
	<i>tag</i>	Received tag. Set to NULL to skip tag processing.
	<i>tagSize</i>	Input size of the received tag to compare with the computed tag, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.

Returns

Status from decrypt operation

## 22.8.4 LTC HASH driver

### 22.8.4.1 Overview

This section describes the programming interface of the LTC HASH driver.

#### Macros

- `#define LTC_HASH_CTX_SIZE 41`  
*LTC HASH Context size.*

#### TypeDefs

- `typedef uint32_t ltc_hash_ctx_t [LTC_HASH_CTX_SIZE]`  
*Storage type used to save hash context.*

#### Enumerations

- `enum ltc_hash_algo_t {`  
 `kLTC_XcbcMac = 0,`  
 `kLTC_Cmac,`  
 `kLTC_Sha1,`  
 `kLTC_Sha224,`  
 `kLTC_Sha256 }`  
*Supported cryptographic block cipher functions for HASH creation.*

#### Functions

- `status_t LTC_HASH_Init (LTC_Type *base, ltc_hash_ctx_t *ctx, ltc_hash_algo_t algo, const uint8_t *key, uint32_t keySize)`  
*Initialize HASH context.*
- `status_t LTC_HASH_Update (ltc_hash_ctx_t *ctx, const uint8_t *input, uint32_t inputSize)`  
*Add data to current HASH.*
- `status_t LTC_HASH_Finish (ltc_hash_ctx_t *ctx, uint8_t *output, uint32_t *outputSize)`  
*Finalize hashing.*
- `status_t LTC_HASH (LTC_Type *base, ltc_hash_algo_t algo, const uint8_t *input, uint32_t inputSize, const uint8_t *key, uint32_t keySize, uint8_t *output, uint32_t *outputSize)`  
*Create HASH on given data.*

## LTC Blocking APIs

### 22.8.4.2 Macro Definition Documentation

#### 22.8.4.2.1 #define LTC\_HASH\_CTX\_SIZE 41

### 22.8.4.3 Typedef Documentation

#### 22.8.4.3.1 typedef uint32\_t ltc\_hash\_ctx\_t[LTC\_HASH\_CTX\_SIZE]

### 22.8.4.4 Enumeration Type Documentation

#### 22.8.4.4.1 enum ltc\_hash\_algo\_t

Enumerator

<i>kLTC_XcbcMac</i>	XCBC-MAC (AES engine)
<i>kLTC_Cmac</i>	CMAC (AES engine)
<i>kLTC_Sha1</i>	SHA_1 (MDHA engine)
<i>kLTC_Sha224</i>	SHA_224 (MDHA engine)
<i>kLTC_Sha256</i>	SHA_256 (MDHA engine)

### 22.8.4.5 Function Documentation

#### 22.8.4.5.1 status\_t LTC\_HASH\_Init ( LTC\_Type \* *base*, ltc\_hash\_ctx\_t \* *ctx*, ltc\_hash\_algo\_t *algo*, const uint8\_t \* *key*, uint32\_t *keySize* )

This function initialize the HASH. Key shall be supplied if the underlaying algorithm is AES XCBC-MAC or CMAC. Key shall be NULL if the underlaying algorithm is SHA.

For XCBC-MAC, the key length must be 16. For CMAC, the key length can be the AES key lengths supported by AES engine. For MDHA the key length argument is ignored.

Parameters

	<i>base</i>	LTC peripheral base address
<i>out</i>	<i>ctx</i>	Output hash context
	<i>algo</i>	Underlaying algorithm to use for hash computation.
	<i>key</i>	Input key (NULL if underlaying algorithm is SHA)
	<i>keySize</i>	Size of input key in bytes

Returns

Status of initialization

**22.8.4.5.2 status\_t LTC\_HASH\_Update ( *ltc\_hash\_ctx\_t \* ctx*, *const uint8\_t \* input*, *uint32\_t inputSize* )**

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed.

## LTC Blocking APIs

Parameters

<i>in, out</i>	<i>ctx</i>	HASH context
	<i>input</i>	Input data
	<i>inputSize</i>	Size of input data in bytes

Returns

Status of the hash update operation

**22.8.4.5.3 status\_t LTC\_HASH\_Finish ( *Ltc\_hash\_ctx\_t \* ctx, uint8\_t \* output, uint32\_t \* outputSize* )**

Outputs the final hash and erases the context.

Parameters

<i>in, out</i>	<i>ctx</i>	Input hash context
<i>out</i>	<i>output</i>	Output hash data
<i>out</i>	<i>outputSize</i>	Output parameter storing the size of the output hash in bytes

Returns

Status of the hash finish operation

**22.8.4.5.4 status\_t LTC\_HASH ( *LTC\_Type \* base, Ltc\_hash\_algo\_t algo, const uint8\_t \* input, uint32\_t inputSize, const uint8\_t \* key, uint32\_t keySize, uint8\_t \* output, uint32\_t \* outputSize* )**

Perform the full keyed HASH in one function call.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>algo</i>	Block cipher algorithm to use for CMAC creation

	<i>input</i>	Input data
	<i>inputSize</i>	Size of input data in bytes
	<i>key</i>	Input key
	<i>keySize</i>	Size of input key in bytes
out	<i>output</i>	Output hash data
out	<i>outputSize</i>	Output parameter storing the size of the output hash in bytes

Returns

Status of the one call hash operation.

## LTC Blocking APIs

### 22.8.5 LTC PKHA driver

#### 22.8.5.1 Overview

This section describes the programming interface of the LTC PKHA driver.

#### Data Structures

- struct [ltc\\_pkha\\_ecc\\_point\\_t](#)  
*PKHA ECC point structure.* [More...](#)

#### Enumerations

- enum [ltc\\_pkha\\_timing\\_t](#) {  
  kLTC\_PKHA\_NoTimingEqualized = 0U,  
  kLTC\_PKHA\_TimingEqualized = 1U }  
*Use of timing equalized version of a PKHA function.*
- enum [ltc\\_pkha\\_f2m\\_t](#) {  
  kLTC\_PKHA\_IntegerArith = 0U,  
  kLTC\_PKHA\_F2mArith = 1U }  
*Integer vs binary polynomial arithmetic selection.*
- enum [ltc\\_pkha\\_montgomery\\_form\\_t](#) {  
  kLTC\_PKHA\_NormalValue = 0U,  
  kLTC\_PKHA\_MontgomeryFormat = 1U }  
*Montgomery or normal PKHA input format.*

#### Functions

- int [LTC\\_PKHA\\_CompareBigNum](#) (const uint8\_t \*a, size\_t sizeA, const uint8\_t \*b, size\_t sizeB)  
*Compare two PKHA big numbers.*
- status\_t [LTC\\_PKHA\\_NormalToMontgomery](#) (LTC\_Type \*base, const uint8\_t \*N, uint16\_t sizeN, uint8\_t \*A, uint16\_t \*sizeA, uint8\_t \*B, uint16\_t \*sizeB, uint8\_t \*R2, uint16\_t \*sizeR2, [ltc\\_pkha\\_timing\\_t](#) equalTime, [ltc\\_pkha\\_f2m\\_t](#) arithType)  
*Converts from integer to Montgomery format.*
- status\_t [LTC\\_PKHA\\_MontgomeryToNormal](#) (LTC\_Type \*base, const uint8\_t \*N, uint16\_t sizeN, uint8\_t \*A, uint16\_t \*sizeA, uint8\_t \*B, uint16\_t \*sizeB, [ltc\\_pkha\\_timing\\_t](#) equalTime, [ltc\\_pkha\\_f2m\\_t](#) arithType)  
*Converts from Montgomery format to int.*
- status\_t [LTC\\_PKHA\\_ModAdd](#) (LTC\_Type \*base, const uint8\_t \*A, uint16\_t sizeA, const uint8\_t \*B, uint16\_t sizeB, const uint8\_t \*N, uint16\_t sizeN, uint8\_t \*result, uint16\_t \*resultSize, [ltc\\_pkha\\_f2m\\_t](#) arithType)  
*Performs modular addition -  $(A + B) \bmod N$ .*
- status\_t [LTC\\_PKHA\\_ModSub1](#) (LTC\_Type \*base, const uint8\_t \*A, uint16\_t sizeA, const uint8\_t \*B, uint16\_t sizeB, const uint8\_t \*N, uint16\_t sizeN, uint8\_t \*result, uint16\_t \*resultSize)  
*Performs modular subtraction -  $(A - B) \bmod N$ .*

- status\_t [LTC\\_PKHA\\_ModSub2](#) (LTC\_Type \*base, const uint8\_t \*A, uint16\_t sizeA, const uint8\_t \*B, uint16\_t sizeB, const uint8\_t \*N, uint16\_t sizeN, uint8\_t \*result, uint16\_t \*resultSize)
 

*Performs modular subtraction -  $(B - A) \bmod N$ .*
- status\_t [LTC\\_PKHA\\_ModMul](#) (LTC\_Type \*base, const uint8\_t \*A, uint16\_t sizeA, const uint8\_t \*B, uint16\_t sizeB, const uint8\_t \*N, uint16\_t sizeN, uint8\_t \*result, uint16\_t \*resultSize, [ltc\\_pkha\\_f2m\\_t](#) arithType, [ltc\\_pkha\\_montgomery\\_form\\_t](#) montIn, [ltc\\_pkha\\_montgomery\\_form\\_t](#) montOut, [ltc\\_pkha\\_timing\\_t](#) equalTime)
 

*Performs modular multiplication -  $(A \times B) \bmod N$ .*
- status\_t [LTC\\_PKHA\\_ModExp](#) (LTC\_Type \*base, const uint8\_t \*A, uint16\_t sizeA, const uint8\_t \*N, uint16\_t sizeN, const uint8\_t \*E, uint16\_t sizeE, uint8\_t \*result, uint16\_t \*resultSize, [ltc\\_pkha\\_f2m\\_t](#) arithType, [ltc\\_pkha\\_montgomery\\_form\\_t](#) montIn, [ltc\\_pkha\\_timing\\_t](#) equalTime)
 

*Performs modular exponentiation -  $(A^E) \bmod N$ .*
- status\_t [LTC\\_PKHA\\_ModRed](#) (LTC\_Type \*base, const uint8\_t \*A, uint16\_t sizeA, const uint8\_t \*N, uint16\_t sizeN, uint8\_t \*result, uint16\_t \*resultSize, [ltc\\_pkha\\_f2m\\_t](#) arithType)
 

*Performs modular reduction -  $(A) \bmod N$ .*
- status\_t [LTC\\_PKHA\\_ModInv](#) (LTC\_Type \*base, const uint8\_t \*A, uint16\_t sizeA, const uint8\_t \*N, uint16\_t sizeN, uint8\_t \*result, uint16\_t \*resultSize, [ltc\\_pkha\\_f2m\\_t](#) arithType)
 

*Performs modular inversion -  $(A^{-1}) \bmod N$ .*
- status\_t [LTC\\_PKHA\\_ModR2](#) (LTC\_Type \*base, const uint8\_t \*N, uint16\_t sizeN, uint8\_t \*result, uint16\_t \*resultSize, [ltc\\_pkha\\_f2m\\_t](#) arithType)
 

*Computes integer Montgomery factor  $R^2 \bmod N$ .*
- status\_t [LTC\\_PKHA\\_GCD](#) (LTC\_Type \*base, const uint8\_t \*A, uint16\_t sizeA, const uint8\_t \*N, uint16\_t sizeN, uint8\_t \*result, uint16\_t \*resultSize, [ltc\\_pkha\\_f2m\\_t](#) arithType)
 

*Calculates the greatest common divisor -  $\text{GCD}(A, N)$ .*
- status\_t [LTC\\_PKHA\\_PrinalityTest](#) (LTC\_Type \*base, const uint8\_t \*A, uint16\_t sizeA, const uint8\_t \*B, uint16\_t sizeB, const uint8\_t \*N, uint16\_t sizeN, bool \*res)
 

*Executes Miller-Rabin primality test.*
- status\_t [LTC\\_PKHA\\_ECC\\_PointAdd](#) (LTC\_Type \*base, const [ltc\\_pkha\\_ecc\\_point\\_t](#) \*A, const [ltc\\_pkha\\_ecc\\_point\\_t](#) \*B, const uint8\_t \*N, const uint8\_t \*R2modN, const uint8\_t \*aCurveParam, const uint8\_t \*bCurveParam, uint8\_t size, [ltc\\_pkha\\_f2m\\_t](#) arithType, [ltc\\_pkha\\_ecc\\_point\\_t](#) \*result)
 

*Adds elliptic curve points -  $A + B$ .*
- status\_t [LTC\\_PKHA\\_ECC\\_PointDouble](#) (LTC\_Type \*base, const [ltc\\_pkha\\_ecc\\_point\\_t](#) \*B, const uint8\_t \*N, const uint8\_t \*aCurveParam, const uint8\_t \*bCurveParam, uint8\_t size, [ltc\\_pkha\\_f2m\\_t](#) arithType, [ltc\\_pkha\\_ecc\\_point\\_t](#) \*result)
 

*Doubles elliptic curve points -  $B + B$ .*
- status\_t [LTC\\_PKHA\\_ECC\\_PointMul](#) (LTC\_Type \*base, const [ltc\\_pkha\\_ecc\\_point\\_t](#) \*A, const uint8\_t \*E, uint8\_t sizeE, const uint8\_t \*N, const uint8\_t \*R2modN, const uint8\_t \*aCurveParam, const uint8\_t \*bCurveParam, uint8\_t size, [ltc\\_pkha\\_timing\\_t](#) equalTime, [ltc\\_pkha\\_f2m\\_t](#) arithType, [ltc\\_pkha\\_ecc\\_point\\_t](#) \*result, bool \*infinity)
 

*Multiples an elliptic curve point by a scalar -  $E \times (A0, A1)$ .*

## LTC Blocking APIs

### 22.8.5.2 Data Structure Documentation

#### 22.8.5.2.1 struct ltc\_pkha\_ecc\_point\_t

##### Data Fields

- `uint8_t * X`  
*X coordinate (affine)*
- `uint8_t * Y`  
*Y coordinate (affine)*

### 22.8.5.3 Enumeration Type Documentation

#### 22.8.5.3.1 enum ltc\_pkha\_timing\_t

Enumerator

*kLTC\_PKHA\_NoTimingEqualized* Normal version of a PKHA operation.

*kLTC\_PKHA\_TimingEqualized* Timing-equalized version of a PKHA operation.

#### 22.8.5.3.2 enum ltc\_pkha\_f2m\_t

Enumerator

*kLTC\_PKHA\_IntegerArith* Use integer arithmetic.

*kLTC\_PKHA\_F2mArith* Use binary polynomial arithmetic.

#### 22.8.5.3.3 enum ltc\_pkha\_montgomery\_form\_t

Enumerator

*kLTC\_PKHA\_NormalValue* PKHA number is normal integer.

*kLTC\_PKHA\_MontgomeryFormat* PKHA number is in montgomery format.

### 22.8.5.4 Function Documentation

#### 22.8.5.4.1 int LTC\_PKHA\_CompareBigNum ( const uint8\_t \* a, size\_t sizeA, const uint8\_t \* b, size\_t sizeB )

Compare two PKHA big numbers. Return 1 for  $a > b$ , -1 for  $a < b$  and 0 if they are same. PKHA big number is lsbyte first. Thus the comparison starts at msbyte which is the last member of tested arrays.

## Parameters

<i>a</i>	First integer represented as an array of bytes, lsbyte first.
<i>sizeA</i>	Size in bytes of the first integer.
<i>b</i>	Second integer represented as an array of bytes, lsbyte first.
<i>sizeB</i>	Size in bytes of the second integer.

## Returns

1 if  $a > b$ .  
-1 if  $a < b$ .  
0 if  $a = b$ .

#### 22.8.5.4.2 `status_t LTC_PKHA_NormalToMontgomery( LTC_Type *base, const uint8_t *N, uint16_t sizeN, uint8_t *A, uint16_t *sizeA, uint8_t *B, uint16_t *sizeB, uint8_t *R2, uint16_t *sizeR2, ltc_pkha_timing_t equalTime, ltc_pkha_f2m_t arithType )`

This function computes  $R2 \bmod N$  and optionally converts A or B into Montgomery format of A or B.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>N</i>	modulus
	<i>sizeN</i>	size of N in bytes
in,out	<i>A</i>	The first input in non-Montgomery format. Output Montgomery format of the first input.
in,out	<i>sizeA</i>	pointer to size variable. On input it holds size of input A in bytes. On output it holds size of Montgomery format of A in bytes.
in,out	<i>B</i>	Second input in non-Montgomery format. Output Montgomery format of the second input.
in,out	<i>sizeB</i>	pointer to size variable. On input it holds size of input B in bytes. On output it holds size of Montgomery format of B in bytes.
out	<i>R2</i>	Output Montgomery factor $R2 \bmod N$ .
out	<i>sizeR2</i>	pointer to size variable. On output it holds size of Montgomery factor $R2 \bmod N$ in bytes.

## LTC Blocking APIs

	<i>equalTime</i>	Run the function time equalized or no timing equalization.
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

**22.8.5.4.3 status\_t LTC\_PKHA\_MontgomeryToNormal ( LTC\_Type \* *base*, const uint8\_t \* *N*, uint16\_t *sizeN*, uint8\_t \* *A*, uint16\_t \* *sizeA*, uint8\_t \* *B*, uint16\_t \* *sizeB*, ltc\_pkha\_timing\_t *equalTime*, ltc\_pkha\_f2m\_t *arithType* )**

This function converts Montgomery format of A or B into int A or B.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>N</i>	modulus.
	<i>sizeN</i>	size of N modulus in bytes.
in,out	<i>A</i>	Input first number in Montgomery format. Output is non-Montgomery format.
in,out	<i>sizeA</i>	pointer to size variable. On input it holds size of the input A in bytes. On output it holds size of non-Montgomery A in bytes.
in,out	<i>B</i>	Input first number in Montgomery format. Output is non-Montgomery format.
in,out	<i>sizeB</i>	pointer to size variable. On input it holds size of the input B in bytes. On output it holds size of non-Montgomery B in bytes.
	<i>equalTime</i>	Run the function time equalized or no timing equalization.
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

**22.8.5.4.4 status\_t LTC\_PKHA\_ModAdd ( LTC\_Type \* *base*, const uint8\_t \* *A*, uint16\_t *sizeA*, const uint8\_t \* *B*, uint16\_t *sizeB*, const uint8\_t \* *N*, uint16\_t *sizeN*, uint8\_t \* *result*, uint16\_t \* *resultSize*, ltc\_pkha\_f2m\_t *arithType* )**

This function performs modular addition of (A + B) mod N, with either integer or binary polynomial (F2m) inputs. In the F2m form, this function is equivalent to a bitwise XOR and it is functionally the same as subtraction.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>A</i>	first addend (integer or binary polynomial)
	<i>sizeA</i>	Size of A in bytes
	<i>B</i>	second addend (integer or binary polynomial)
	<i>sizeB</i>	Size of B in bytes
	<i>N</i>	modulus. For F2m operation this can be NULL, as N is ignored during F2m polynomial addition.
	<i>sizeN</i>	Size of N in bytes. This must be given for both integer and F2m polynomial additions.
out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)

## Returns

Operation status.

**22.8.5.4.5 status\_t LTC\_PKHA\_ModSub1 ( LTC\_Type \* *base*, const uint8\_t \* *A*, uint16\_t *sizeA*, const uint8\_t \* *B*, uint16\_t *sizeB*, const uint8\_t \* *N*, uint16\_t *sizeN*, uint8\_t \* *result*, uint16\_t \* *resultSize* )**

This function performs modular subtraction of (A - B) mod N with integer inputs.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>A</i>	first addend (integer or binary polynomial)
	<i>sizeA</i>	Size of A in bytes
	<i>B</i>	second addend (integer or binary polynomial)
	<i>sizeB</i>	Size of B in bytes
	<i>N</i>	modulus

## LTC Blocking APIs

	<i>sizeN</i>	Size of N in bytes
out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes

Returns

Operation status.

**22.8.5.4.6 `status_t LTC_PKHA_ModSub2 ( LTC_Type * base, const uint8_t * A, uint16_t sizeA, const uint8_t * B, uint16_t sizeB, const uint8_t * N, uint16_t sizeN, uint8_t * result, uint16_t * resultSize )`**

This function performs modular subtraction of (B - A) mod N, with integer inputs.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>A</i>	first addend (integer or binary polynomial)
	<i>sizeA</i>	Size of A in bytes
	<i>B</i>	second addend (integer or binary polynomial)
	<i>sizeB</i>	Size of B in bytes
	<i>N</i>	modulus
	<i>sizeN</i>	Size of N in bytes
out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes

Returns

Operation status.

**22.8.5.4.7 `status_t LTC_PKHA_ModMul ( LTC_Type * base, const uint8_t * A, uint16_t sizeA, const uint8_t * B, uint16_t sizeB, const uint8_t * N, uint16_t sizeN, uint8_t * result, uint16_t * resultSize, ltc_pkha_f2m_t arithType, ltc_pkha_montgomery_form_t montIn, ltc_pkha_montgomery_form_t montOut, ltc_pkha_timing_t equalTime )`**

This function performs modular multiplication with either integer or binary polynomial (F2m) inputs. It can optionally specify whether inputs and/or outputs will be in Montgomery form or not.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>A</i>	first addend (integer or binary polynomial)
	<i>sizeA</i>	Size of A in bytes
	<i>B</i>	second addend (integer or binary polynomial)
	<i>sizeB</i>	Size of B in bytes
	<i>N</i>	modulus.
	<i>sizeN</i>	Size of N in bytes
out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)
	<i>montIn</i>	Format of inputs
	<i>montOut</i>	Format of output
	<i>equalTime</i>	Run the function time equalized or no timing equalization. This argument is ignored for F2m modular multiplication.

## Returns

Operation status.

**22.8.5.4.8 status\_t LTC\_PKHA\_ModExp ( LTC\_Type \* *base*, const uint8\_t \* *A*, uint16\_t *sizeA*, const uint8\_t \* *N*, uint16\_t *sizeN*, const uint8\_t \* *E*, uint16\_t *sizeE*, uint8\_t \* *result*, uint16\_t \* *resultSize*, ltc\_pkha\_f2m\_t *arithType*, ltc\_pkha\_montgomery\_form\_t *montIn*, ltc\_pkha\_timing\_t *equalTime* )**

This function performs modular exponentiation with either integer or binary polynomial (F2m) inputs.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>A</i>	first addend (integer or binary polynomial)
	<i>sizeA</i>	Size of A in bytes

## LTC Blocking APIs

	<i>N</i>	modulus
	<i>sizeN</i>	Size of N in bytes
	<i>E</i>	exponent
	<i>sizeE</i>	Size of E in bytes
out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes
	<i>montIn</i>	Format of A input (normal or Montgomery)
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)
	<i>equalTime</i>	Run the function time equalized or no timing equalization.

Returns

Operation status.

**22.8.5.4.9 `status_t LTC_PKHA_ModRed ( LTC_Type * base, const uint8_t * A, uint16_t sizeA, const uint8_t * N, uint16_t sizeN, uint8_t * result, uint16_t * resultSize, ltc_pkha_f2m_t arithType )`**

This function performs modular reduction with either integer or binary polynomial (F2m) inputs.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>A</i>	first addend (integer or binary polynomial)
	<i>sizeA</i>	Size of A in bytes
	<i>N</i>	modulus
	<i>sizeN</i>	Size of N in bytes
out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

**22.8.5.4.10 status\_t LTC\_PKHA\_ModInv ( LTC\_Type \* *base*, const uint8\_t \* *A*, uint16\_t *sizeA*, const uint8\_t \* *N*, uint16\_t *sizeN*, uint8\_t \* *result*, uint16\_t \* *resultSize*, ltc\_pkha\_f2m\_t *arithType* )**

This function performs modular inversion with either integer or binary polynomial (F2m) inputs.

## LTC Blocking APIs

Parameters

	<i>base</i>	LTC peripheral base address
	<i>A</i>	first addend (integer or binary polynomial)
	<i>sizeA</i>	Size of A in bytes
	<i>N</i>	modulus
	<i>sizeN</i>	Size of N in bytes
out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

**22.8.5.4.11 status\_t LTC\_PKHA\_ModR2 ( LTC\_Type \* *base*, const uint8\_t \* *N*, uint16\_t *sizeN*, uint8\_t \* *result*, uint16\_t \* *resultSize*, ltc\_pkha\_f2m\_t *arithType* )**

This function computes a constant to assist in converting operands into the Montgomery residue system representation.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>N</i>	modulus
	<i>sizeN</i>	Size of N in bytes
out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

**22.8.5.4.12 status\_t LTC\_PKHA\_GCD ( LTC\_Type \* *base*, const uint8\_t \* *A*, uint16\_t *sizeA*, const uint8\_t \* *N*, uint16\_t *sizeN*, uint8\_t \* *result*, uint16\_t \* *resultSize*, ltc\_pkha\_f2m\_t *arithType* )**

This function calculates the greatest common divisor of two inputs with either integer or binary polynomial (F2m) inputs.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>A</i>	first value (must be smaller than or equal to N)
	<i>sizeA</i>	Size of A in bytes
	<i>N</i>	second value (must be non-zero)
	<i>sizeN</i>	Size of N in bytes
out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

**22.8.5.4.13 `status_t LTC_PKHA_PrinalityTest ( LTC_Type * base, const uint8_t * A, uint16_t sizeA, const uint8_t * B, uint16_t sizeB, const uint8_t * N, uint16_t sizeN, bool * res )`**

This function calculates whether or not a candidate prime number is likely to be a prime.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>A</i>	initial random seed
	<i>sizeA</i>	Size of A in bytes
	<i>B</i>	number of trial runs
	<i>sizeB</i>	Size of B in bytes
	<i>N</i>	candidate prime integer
	<i>sizeN</i>	Size of N in bytes
out	<i>res</i>	True if the value is likely prime or false otherwise

Returns

Operation status.

## LTC Blocking APIs

```
22.8.5.4.14 status_t LTC_PKHA_ECC_PointAdd( LTC_Type * base, const ltc_pkha_ecc_point_t  
* A, const ltc_pkha_ecc_point_t * B, const uint8_t * N, const uint8_t * R2modN,  
const uint8_t * aCurveParam, const uint8_t * bCurveParam, uint8_t size,  
ltc_pkha_f2m_t arithType, ltc_pkha_ecc_point_t * result )
```

This function performs ECC point addition over a prime field (Fp) or binary field (F2m) using affine coordinates.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>A</i>	Left-hand point
	<i>B</i>	Right-hand point
	<i>N</i>	Prime modulus of the field
	<i>R2modN</i>	NULL (the function computes R2modN internally) or pointer to pre-computed R2modN (obtained from <a href="#">LTC_PKHA_ModR2()</a> function).
	<i>aCurveParam</i>	A parameter from curve equation
	<i>bCurveParam</i>	B parameter from curve equation (constant)
	<i>size</i>	Size in bytes of curve points and parameters
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)
out	<i>result</i>	Result point

## Returns

Operation status.

```
22.8.5.4.15 status_t LTC_PKHA_ECC_PointDouble ( LTC_Type * base, const
                                              ltc_pkha_ecc_point_t * B, const uint8_t * N, const uint8_t * aCurveParam,
                                              const uint8_t * bCurveParam, uint8_t size, ltc_pkha_f2m_t arithType,
                                              ltc_pkha_ecc_point_t * result )
```

This function performs ECC point doubling over a prime field (Fp) or binary field (F2m) using affine coordinates.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>B</i>	Point to double
	<i>N</i>	Prime modulus of the field
	<i>aCurveParam</i>	A parameter from curve equation
	<i>bCurveParam</i>	B parameter from curve equation (constant)

## LTC Blocking APIs

	<i>size</i>	Size in bytes of curve points and parameters
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)
out	<i>result</i>	Result point

Returns

Operation status.

```
22.8.5.4.16 status_t LTC_PKHA_ECC_PointMul ( LTC_Type * base, const ltc_pkha_ecc_point_t
* A, const uint8_t * E, uint8_t sizeE, const uint8_t * N, const uint8_t * R2modN,
const uint8_t * aCurveParam, const uint8_t * bCurveParam, uint8_t size,
ltc_pkha_timing_t equalTime, ltc_pkha_f2m_t arithType, ltc_pkha_ecc_point_t * result, bool * infinity )
```

This function performs ECC point multiplication to multiply an ECC point by a scalar integer multiplier over a prime field (Fp) or a binary field (F2m).

Parameters

	<i>base</i>	LTC peripheral base address
	<i>A</i>	Point as multiplicand
	<i>E</i>	Scalar multiple
	<i>sizeE</i>	The size of E, in bytes
	<i>N</i>	Modulus, a prime number for the Fp field or Irreducible polynomial for F2m field.
	<i>R2modN</i>	NULL (the function computes R2modN internally) or pointer to pre-computed R2modN (obtained from <a href="#">LTC_PKHA_ModR2()</a> function).
	<i>aCurveParam</i>	A parameter from curve equation
	<i>bCurveParam</i>	B parameter from curve equation (C parameter for operation over F2m).
	<i>size</i>	Size in bytes of curve points and parameters
	<i>equalTime</i>	Run the function time equalized or no timing equalization.
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)

out	<i>result</i>	Result point
out	<i>infinity</i>	Output true if the result is point of infinity, and false otherwise. Writing of this output will be ignored if the argument is NULL.

Returns

Operation status.

## LTC Non-blocking eDMA APIs

### 22.9 LTC Non-blocking eDMA APIs

#### 22.9.1 Overview

This section describes the programming interface of the LTC eDMA Non Blocking functions

## Modules

- [LTC eDMA AES driver](#)
- [LTC eDMA DES driver](#)

## Data Structures

- struct [ltc\\_edma\\_handle\\_t](#)  
*LTC eDMA handle.* [More...](#)

## Typedefs

- [typedef void\(\\* ltc\\_edma\\_callback\\_t \)\(LTC\\_Type \\*base, ltc\\_edma\\_handle\\_t \\*handle, status\\_t status, void \\*userData\)](#)  
*LTC eDMA callback function.*
- [typedef status\\_t\(\\* ltc\\_edma\\_state\\_machine\\_t \)\(LTC\\_Type \\*base, ltc\\_edma\\_handle\\_t \\*handle\)](#)  
*LTC eDMA state machine function.*

## Functions

- [void LTC\\_CreateHandleEDMA \(LTC\\_Type \\*base, ltc\\_edma\\_handle\\_t \\*handle, ltc\\_edma\\_callback\\_t callback, void \\*userData, edma\\_handle\\_t \\*inputFifoEdmaHandle, edma\\_handle\\_t \\*outputFifoEdmaHandle\)](#)  
*Init the LTC eDMA handle which is used in transnational functions.*

#### 22.9.2 Data Structure Documentation

##### 22.9.2.1 struct \_ltc\_edma\_handle

It is defined only for private usage inside LTC eDMA driver.

## Data Fields

- [ltc\\_edma\\_callback\\_t callback](#)  
*Callback function.*
- [void \\* userData](#)

*LTC callback function parameter.*

- **edma\_handle\_t \* inputFifoEdmaHandle**  
*The eDMA TX channel used.*
- **edma\_handle\_t \* outputFifoEdmaHandle**  
*The eDMA RX channel used.*
- **ltc\_edma\_state\_machine\_t state\_machine**  
*State machine.*
- **uint32\_t state**  
*Internal state.*
- **const uint8\_t \* inData**  
*Input data.*
- **uint8\_t \* outData**  
*Output data.*
- **uint32\_t size**  
*Size of input and output data in bytes.*
- **uint32\_t modeReg**  
*LTC mode register.*
- **uint8\_t \* counter**  
*Input counter (updates on return)*
- **const uint8\_t \* key**  
*Input key to use for forward AES cipher.*
- **uint32\_t keySize**  
*Size of the input key, in bytes.*
- **uint8\_t \* counterlast**  
*Output cipher of last counter, for chained CTR calls.*
- **uint32\_t \* szLeft**  
*Output number of bytes in left unused in counterlast block.*
- **uint32\_t lastSize**  
*Last size.*

## LTC Non-blocking eDMA APIs

### 22.9.2.1.0.1 Field Documentation

22.9.2.1.0.1.1 `ltc_edma_callback_t ltc_edma_handle_t::callback`

22.9.2.1.0.1.2 `void* ltc_edma_handle_t::userData`

22.9.2.1.0.1.3 `edma_handle_t* ltc_edma_handle_t::inputFifoEdmaHandle`

22.9.2.1.0.1.4 `edma_handle_t* ltc_edma_handle_t::outputFifoEdmaHandle`

22.9.2.1.0.1.5 `ltc_edma_state_machine_t ltc_edma_handle_t::state_machine`

22.9.2.1.0.1.6 `uint32_t ltc_edma_handle_t::state`

22.9.2.1.0.1.7 `const uint8_t* ltc_edma_handle_t::inData`

22.9.2.1.0.1.8 `uint8_t* ltc_edma_handle_t::outData`

22.9.2.1.0.1.9 `uint32_t ltc_edma_handle_t::size`

22.9.2.1.0.1.10 `uint32_t ltc_edma_handle_t::modeReg`

22.9.2.1.0.1.11 `uint32_t ltc_edma_handle_t::keySize`

Must be 16, 24, or 32.

22.9.2.1.0.1.12 `uint8_t* ltc_edma_handle_t::counterlast`

NULL can be passed if chained calls are not used.

22.9.2.1.0.1.13 `uint32_t* ltc_edma_handle_t::szLeft`

NULL can be passed if chained calls are not used.

22.9.2.1.0.1.14 `uint32_t ltc_edma_handle_t::lastSize`

### 22.9.3 Typedef Documentation

22.9.3.1 `typedef void(* ltc_edma_callback_t)(LTC_Type *base, ltc_edma_handle_t *handle, status_t status, void *userData)`

22.9.3.2 `typedef status_t(* ltc_edma_state_machine_t)(LTC_Type *base, ltc_edma_handle_t *handle)`

It is defined only for private usage inside LTC eDMA driver.

## 22.9.4 Function Documentation

**22.9.4.1 void LTC\_CreateHandleEDMA ( LTC\_Type \* *base*, ltc\_edma\_handle\_t \* *handle*, ltc\_edma\_callback\_t *callback*, void \* *userData*, edma\_handle\_t \* *inputFifoEdmaHandle*, edma\_handle\_t \* *outputFifoEdmaHandle* )**

## LTC Non-blocking eDMA APIs

Parameters

<i>base</i>	LTC module base address
<i>handle</i>	Pointer to ltc_edma_handle_t structure
<i>callback</i>	Callback function, NULL means no callback.
<i>userData</i>	Callback function parameter.
<i>inputFifo-EdmaHandle</i>	User requested eDMA handle for Input FIFO eDMA.
<i>outputFifo-EdmaHandle</i>	User requested eDMA handle for Output FIFO eDMA.

## 22.9.5 LTC eDMA DES driver

### 22.9.5.1 Overview

This section describes the programming interface of the LTC eDMA DES driver.

### Functions

- status\_t [LTC\\_DES\\_EncryptEcbEDMA](#) (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t key[LTC\_DES\_KEY\_SIZE])  
*Encrypts DES using ECB block mode.*
- status\_t [LTC\\_DES\\_DecryptEcbEDMA](#) (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t key[LTC\_DES\_KEY\_SIZE])  
*Decrypts DES using ECB block mode.*
- status\_t [LTC\\_DES\\_EncryptCbcEDMA](#) (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key[LTC\_DES\_KEY\_SIZE])  
*Encrypts DES using CBC block mode.*
- status\_t [LTC\\_DES\\_DecryptCbcEDMA](#) (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key[LTC\_DES\_KEY\_SIZE])  
*Decrypts DES using CBC block mode.*
- status\_t [LTC\\_DES\\_EncryptCfbEDMA](#) (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key[LTC\_DES\_KEY\_SIZE])  
*Encrypts DES using CFB block mode.*
- status\_t [LTC\\_DES\\_DecryptCfbEDMA](#) (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key[LTC\_DES\_KEY\_SIZE])  
*Decrypts DES using CFB block mode.*
- status\_t [LTC\\_DES\\_EncryptOfbEDMA](#) (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key[LTC\_DES\_KEY\_SIZE])  
*Encrypts DES using OFB block mode.*
- status\_t [LTC\\_DES\\_DecryptOfbEDMA](#) (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key[LTC\_DES\_KEY\_SIZE])  
*Decrypts DES using OFB block mode.*
- status\_t [LTC\\_DES2\\_EncryptEcbEDMA](#) (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE])  
*Encrypts triple DES using ECB block mode with two keys.*
- status\_t [LTC\\_DES2\\_DecryptEcbEDMA](#) (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE])  
*Decrypts triple DES using ECB block mode with two keys.*
- status\_t [LTC\\_DES2\\_EncryptCbcEDMA](#) (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const

## LTC Non-blocking eDMA APIs

```
uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const  
uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE])
```

*Encrypts triple DES using CBC block mode with two keys.*

- status\_t **LTC\_DES2\_DecryptCbcEDMA** (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const  
uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const  
uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE])

*Decrypts triple DES using CBC block mode with two keys.*

- status\_t **LTC\_DES2\_EncryptCfbEDMA** (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const  
uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const  
uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE])

*Encrypts triple DES using CFB block mode with two keys.*

- status\_t **LTC\_DES2\_DecryptCfbEDMA** (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const  
uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const  
uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE])

*Decrypts triple DES using CFB block mode with two keys.*

- status\_t **LTC\_DES2\_EncryptOfbEDMA** (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const  
uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const  
uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE])

*Encrypts triple DES using OFB block mode with two keys.*

- status\_t **LTC\_DES2\_DecryptOfbEDMA** (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const  
uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const  
uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE])

*Decrypts triple DES using OFB block mode with two keys.*

- status\_t **LTC\_DES3\_EncryptEcbEDMA** (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const  
uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t key1[LTC\_DES\_KEY\_SIZE],  
const uint8\_t key2[LTC\_DES\_KEY\_SIZE], const uint8\_t key3[LTC\_DES\_KEY\_SIZE])

*Encrypts triple DES using ECB block mode with three keys.*

- status\_t **LTC\_DES3\_DecryptEcbEDMA** (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const  
uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t key1[LTC\_DES\_KEY\_SIZE],  
const uint8\_t key2[LTC\_DES\_KEY\_SIZE], const uint8\_t key3[LTC\_DES\_KEY\_SIZE])

*Decrypts triple DES using ECB block mode with three keys.*

- status\_t **LTC\_DES3\_EncryptCbcEDMA** (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const  
uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const  
uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE], const uint8\_t  
key3[LTC\_DES\_KEY\_SIZE])

*Encrypts triple DES using CBC block mode with three keys.*

- status\_t **LTC\_DES3\_DecryptCbcEDMA** (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const  
uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const  
uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE], const uint8\_t  
key3[LTC\_DES\_KEY\_SIZE])

*Decrypts triple DES using CBC block mode with three keys.*

- status\_t **LTC\_DES3\_EncryptCfbEDMA** (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const

```
uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const
uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t
key3[LTC_DES_KEY_SIZE])
```

*Decrypts triple DES using CFB block mode with three keys.*

- status\_t **LTC\_DES3\_EncryptOfbEDMA** (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE], const uint8\_t key3[LTC\_DES\_KEY\_SIZE])

*Encrypts triple DES using OFB block mode with three keys.*

- status\_t **LTC\_DES3\_DecryptOfbEDMA** (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[LTC\_DES\_IV\_SIZE], const uint8\_t key1[LTC\_DES\_KEY\_SIZE], const uint8\_t key2[LTC\_DES\_KEY\_SIZE], const uint8\_t key3[LTC\_DES\_KEY\_SIZE])

*Decrypts triple DES using OFB block mode with three keys.*

## 22.9.5.2 Function Documentation

### 22.9.5.2.1 status\_t LTC\_DES\_EncryptEcbEDMA ( LTC\_Type \* *base*, ltc\_edma\_handle\_t \* *handle*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *key*[LTC\_DES\_KEY\_SIZE] )

Encrypts DES using ECB block mode.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key</i>	Input key to use for encryption

Returns

Status from encrypt/decrypt operation

### 22.9.5.2.2 status\_t LTC\_DES\_DecryptEcbEDMA ( LTC\_Type \* *base*, ltc\_edma\_handle\_t \* *handle*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t *key*[LTC\_DES\_KEY\_SIZE] )

Decrypts DES using ECB block mode.

## LTC Non-blocking eDMA APIs

Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key</i>	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

**22.9.5.2.3 status\_t LTC\_DES\_EncryptCbcEDMA ( LTC\_Type \* *base*, ltc\_edma\_handle\_t \* *handle*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key*[LTC\_DES\_KEY\_SIZE] )**

Encrypts DES using CBC block mode.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Ouput ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key</i>	Input key to use for encryption

Returns

Status from encrypt/decrypt operation

**22.9.5.2.4 status\_t LTC\_DES\_DecryptCbcEDMA ( LTC\_Type \* *base*, ltc\_edma\_handle\_t \* *handle*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key*[LTC\_DES\_KEY\_SIZE] )**

Decrypts DES using CBC block mode.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key</i>	Input key to use for decryption

## Returns

Status from encrypt/decrypt operation

```
22.9.5.2.5 status_t LTC_DES_EncryptCfbEDMA ( LTC_Type * base, ltc_edma_handle_t *  
    handle, const uint8_t * plaintext, uint8_t * ciphertext, uint32_t size, const uint8_t  
    iv[LTC_DES_IV_SIZE], const uint8_t key[LTC_DES_KEY_SIZE] )
```

Encrypts DES using CFB block mode.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>plaintext</i>	Input plaintext to encrypt
	<i>size</i>	Size of input data in bytes
	<i>iv</i>	Input initial block.
	<i>key</i>	Input key to use for encryption
out	<i>ciphertext</i>	Output ciphertext

## Returns

Status from encrypt/decrypt operation

## LTC Non-blocking eDMA APIs

22.9.5.2.6 `status_t LTC_DES_DecryptCfbEDMA ( LTC_Type * base, Itc_edma_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key[LTC_DES_KEY_SIZE] )`

Decrypts DES using CFB block mode.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial block.
	<i>key</i>	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

**22.9.5.2.7 status\_t LTC\_DES\_EncryptOfbEDMA ( LTC\_Type \* *base*, ltc\_edma\_handle\_t \* *handle*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key*[LTC\_DES\_KEY\_SIZE] )**

Encrypts DES using OFB block mode.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key</i>	Input key to use for encryption

Returns

Status from encrypt/decrypt operation

## LTC Non-blocking eDMA APIs

```
22.9.5.2.8 status_t LTC_DES_DecryptOfbEDMA ( LTC_Type * base, ltc_edma_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key[LTC_DES_KEY_SIZE] )
```

Decrypts DES using OFB block mode.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to <code>ltc_edma_handle_t</code> structure which stores the transaction state.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key</i>	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

**22.9.5.2.9 `status_t LTC_DES2_EncryptEcbEDMA ( LTC_Type * base, ltc_edma_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, uint32_t size, const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE] )`**

Encrypts triple DES using ECB block mode with two keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to <code>ltc_edma_handle_t</code> structure which stores the transaction state.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

## LTC Non-blocking eDMA APIs

22.9.5.2.10 `status_t LTC_DES2_DecryptEcbEDMA ( LTC_Type * base, ltc_edma_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, uint32_t size, const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE] )`

Decrypts triple DES using ECB block mode with two keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

**22.9.5.2.11 status\_t LTC\_DES2\_EncryptCbcEDMA ( LTC\_Type \* *base*, ltc\_edma\_handle\_t \* *handle*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE] )**

Encrypts triple DES using CBC block mode with two keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key1</i>	First input key for key bundle

## LTC Non-blocking eDMA APIs

	<i>key2</i>	Second input key for key bundle
--	-------------	---------------------------------

Returns

Status from encrypt/decrypt operation

```
22.9.5.2.12 status_t LTC_DES2_DecryptCbcEDMA ( LTC_Type * base, ltc_edma_handle_t *  
    handle, const uint8_t * ciphertext, uint8_t * plaintext, uint32_t size, const uint8_t  
    iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t  
    key2[LTC_DES_KEY_SIZE] )
```

Decrypts triple DES using CBC block mode with two keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

```
22.9.5.2.13 status_t LTC_DES2_EncryptCfbEDMA ( LTC_Type * base, ltc_edma_handle_t *  
    handle, const uint8_t * plaintext, uint8_t * ciphertext, uint32_t size, const uint8_t  
    iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t  
    key2[LTC_DES_KEY_SIZE] )
```

Encrypts triple DES using CFB block mode with two keys.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial block.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

## Returns

Status from encrypt/decrypt operation

**22.9.5.2.14 status\_t LTC\_DES2\_DecryptCfbEDMA ( LTC\_Type \* *base*, ltc\_edma\_handle\_t \* *handle*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t \* *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE] )**

Decrypts triple DES using CFB block mode with two keys.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial block.
	<i>key1</i>	First input key for key bundle

## LTC Non-blocking eDMA APIs

	<i>key2</i>	Second input key for key bundle
--	-------------	---------------------------------

Returns

Status from encrypt/decrypt operation

**22.9.5.2.15 status\_t LTC\_DES2\_EncryptOfbEDMA ( LTC\_Type \* *base*, ltc\_edma\_handle\_t \* *handle*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE] )**

Encrypts triple DES using OFB block mode with two keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

**22.9.5.2.16 status\_t LTC\_DES2\_DecryptOfbEDMA ( LTC\_Type \* *base*, ltc\_edma\_handle\_t \* *handle*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE] )**

Decrypts triple DES using OFB block mode with two keys.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

## Returns

Status from encrypt/decrypt operation

```
22.9.5.2.17 status_t LTC_DES3_EncryptEcbEDMA ( LTC_Type * base, ltc_edma_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, uint32_t size, const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE] )
```

Encrypts triple DES using ECB block mode with three keys.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key1</i>	First input key for key bundle

## LTC Non-blocking eDMA APIs

	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

**22.9.5.2.18 status\_t LTC\_DES3\_DecryptEcbEDMA ( LTC\_Type \* *base*, ltc\_edma\_handle\_t \* *handle*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key3*[LTC\_DES\_KEY\_SIZE] )**

Decrypts triple DES using ECB block mode with three keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

**22.9.5.2.19 status\_t LTC\_DES3\_EncryptCbcEDMA ( LTC\_Type \* *base*, ltc\_edma\_handle\_t \* *handle*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key3*[LTC\_DES\_KEY\_SIZE] )**

Encrypts triple DES using CBC block mode with three keys.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

## Returns

Status from encrypt/decrypt operation

```
22.9.5.2.20 status_t LTC_DES3_DecryptCbcEDMA ( LTC_Type * base, ltc_edma_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, uint32_t size, const uint8_t _t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t _t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE] )
```

Decrypts triple DES using CBC block mode with three keys.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.

## LTC Non-blocking eDMA APIs

	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

**22.9.5.2.21 status\_t LTC\_DES3\_EncryptCfbEDMA ( LTC\_Type \* *base*, ltc\_edma\_handle\_t \* *handle*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t \* *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key3*[LTC\_DES\_KEY\_SIZE] )**

Encrypts triple DES using CFB block mode with three keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and ouput data in bytes
	<i>iv</i>	Input initial block.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

**22.9.5.2.22 status\_t LTC\_DES3\_DecryptCfbEDMA ( LTC\_Type \* *base*, ltc\_edma\_handle\_t \* *handle*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t \* *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key3*[LTC\_DES\_KEY\_SIZE] )**

Decrypts triple DES using CFB block mode with three keys.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input data in bytes
	<i>iv</i>	Input initial block.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

## Returns

Status from encrypt/decrypt operation

**22.9.5.2.23 status\_t LTC\_DES3\_EncryptOfbEDMA ( LTC\_Type \* *base*, ltc\_edma\_handle\_t \* *handle*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, uint32\_t *size*, const uint8\_t \* *iv*[LTC\_DES\_IV\_SIZE], const uint8\_t *key1*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key2*[LTC\_DES\_KEY\_SIZE], const uint8\_t *key3*[LTC\_DES\_KEY\_SIZE] )**

Encrypts triple DES using OFB block mode with three keys.

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.

## LTC Non-blocking eDMA APIs

	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
22.9.5.2.24 status_t LTC_DES3_DecryptOfbEDMA ( LTC_Type * base, ltc_edma_handle_t *  
    handle, const uint8_t * ciphertext, uint8_t * plaintext, uint32_t size, const uint8_t *  
    iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE] )
```

Decrypts triple DES using OFB block mode with three keys.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

## 22.9.6 LTC eDMA AES driver

### 22.9.6.1 Overview

This section describes the programming interface of the LTC eDMA AES driver.

#### Macros

- #define [LTC\\_AES\\_DecryptCtrEDMA](#)(base, handle, input, output, size, counter, key, keySize, counterlast, szLeft) [LTC\\_AES\\_CryptCtrEDMA](#)(base, handle, input, output, size, counter, key, keySize, counterlast, szLeft)  
*AES CTR decrypt is mapped to the AES CTR generic operation.*
- #define [LTC\\_AES\\_EncryptCtrEDMA](#)(base, handle, input, output, size, counter, key, keySize, counterlast, szLeft) [LTC\\_AES\\_CryptCtrEDMA](#)(base, handle, input, output, size, counter, key, keySize, counterlast, szLeft)  
*AES CTR encrypt is mapped to the AES CTR generic operation.*

#### Functions

- status\_t [LTC\\_AES\\_EncryptEcbEDMA](#) (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t \*key, uint32\_t keySize)  
*Encrypts AES using the ECB block mode.*
- status\_t [LTC\\_AES\\_DecryptEcbEDMA](#) (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t \*key, uint32\_t keySize, [ltc\\_aes\\_key\\_t](#) keyType)  
*Decrypts AES using ECB block mode.*
- status\_t [LTC\\_AES\\_EncryptCbcEDMA](#) (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const uint8\_t \*plaintext, uint8\_t \*ciphertext, uint32\_t size, const uint8\_t iv[[LTC\\_AES\\_IV\\_SIZE](#)], const uint8\_t \*key, uint32\_t keySize)  
*Encrypts AES using CBC block mode.*
- status\_t [LTC\\_AES\\_DecryptCbcEDMA](#) (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const uint8\_t \*ciphertext, uint8\_t \*plaintext, uint32\_t size, const uint8\_t iv[[LTC\\_AES\\_IV\\_SIZE](#)], const uint8\_t \*key, uint32\_t keySize, [ltc\\_aes\\_key\\_t](#) keyType)  
*Decrypts AES using CBC block mode.*
- status\_t [LTC\\_AES\\_CryptCtrEDMA](#) (LTC\_Type \*base, ltc\_edma\_handle\_t \*handle, const uint8\_t \*input, uint8\_t \*output, uint32\_t size, uint8\_t counter[[LTC\\_AES\\_BLOCK\\_SIZE](#)], const uint8\_t \*key, uint32\_t keySize, uint8\_t counterlast[[LTC\\_AES\\_BLOCK\\_SIZE](#)], uint32\_t \*szLeft)  
*Encrypts or decrypts AES using CTR block mode.*

## LTC Non-blocking eDMA APIs

### 22.9.6.2 Function Documentation

22.9.6.2.1 `status_t LTC_AES_EncryptEcbEDMA ( LTC_Type * base, ltc_edma_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, uint32_t size, const uint8_t * key, uint32_t keySize )`

Encrypts AES using the ECB block mode.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>plaintext</i>	Input plain text to encrypt
out	<i>ciphertext</i>	Output cipher text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>key</i>	Input key to use for encryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from encrypt operation

**22.9.6.2.2 status\_t LTC\_AES\_DecryptEcbEDMA ( LTC\_Type \* *base*, ltc\_edma\_handle\_t \* *handle*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, uint32\_t *size*, const uint8\_t \* *key*, uint32\_t *keySize*, ltc\_aes\_key\_t *keyType* )**

Decrypts AES using ECB block mode.

Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>ciphertext</i>	Input cipher text to decrypt
out	<i>plaintext</i>	Output plain text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>key</i>	Input key.
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
	<i>keyType</i>	Input type of the key (allows to directly load decrypt key for AES ECB decrypt operation.)

Returns

Status from decrypt operation

## LTC Non-blocking eDMA APIs

22.9.6.2.3 `status_t LTC_AES_EncryptCbcEDMA ( LTC_Type * base, ltc_edma_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, uint32_t size, const uint8_t * iv[LTC_AES_IV_SIZE], const uint8_t * key, uint32_t keySize )`

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>plaintext</i>	Input plain text to encrypt
out	<i>ciphertext</i>	Output cipher text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>iv</i>	Input initial vector to combine with the first input block.
	<i>key</i>	Input key to use for encryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.

## Returns

Status from encrypt operation

```
22.9.6.2.4 status_t LTC_AES_DecryptCbcEDMA ( LTC_Type * base, ltc_edma_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, uint32_t size, const uint8_t iv[LTC_AES_IV_SIZE], const uint8_t * key, uint32_t keySize, ltc_aes_key_t keyType )
```

## Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>ciphertext</i>	Input cipher text to decrypt
out	<i>plaintext</i>	Output plain text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>iv</i>	Input initial vector to combine with the first input block.
	<i>key</i>	Input key to use for decryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
	<i>keyType</i>	Input type of the key (allows to directly load decrypt key for AES CBC decrypt operation.)

## Returns

Status from decrypt operation

## LTC Non-blocking eDMA APIs

```
22.9.6.2.5 status_t LTC_AES_CryptCtrEDMA ( LTC_Type * base, ltc_edma_handle_t  
* handle, const uint8_t * input, uint8_t * output, uint32_t size, uint8_t  
counter[LTC_AES_BLOCK_SIZE], const uint8_t * key, uint32_t keySize, uint8_t  
counterlast[LTC_AES_BLOCK_SIZE], uint32_t * szLeft )
```

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

### Parameters

	<i>base</i>	LTC peripheral base address
	<i>handle</i>	pointer to ltc_edma_handle_t structure which stores the transaction state.
	<i>input</i>	Input data for CTR block mode
out	<i>output</i>	Output data for CTR block mode
	<i>size</i>	Size of input and output data in bytes
in,out	<i>counter</i>	Input counter (updates on return)
	<i>key</i>	Input key to use for forward AES cipher
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
out	<i>counterlast</i>	Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
out	<i>szLeft</i>	Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.

### Returns

Status from encrypt operation

# Chapter 23

## MPU: Memory Protection Unit

### 23.1 Overview

The MPU driver provides hardware access control for all memory references generated in the device. Use the MPU driver to program the region descriptors that define memory spaces and their access rights. After initialization, the MPU concurrently monitors the system bus transactions and evaluates the appropriateness.

### 23.2 Initialization and Deinitialize

To initialize the MPU module, call the [MPU\\_Init\(\)](#) function and provide the user configuration data structure. This function sets the configuration of the MPU module automatically and enables the MPU module.

Note that the configuration start address, end address, the region valid value, and the debugger's access permission for the MPU region 0 cannot be changed.

This is example code to configure the MPU driver:

```
// Defines the MPU memory access permission configuration structure. //
mpu_rwxrights_master_access_control_t mpuRwxAccessRightsMasters =
{
    kMPU_SupervisorReadWriteExecute,
    kMPU_UserNoAccessRights,
    kMPU_IdentifierDisable,
    kMPU_SupervisorEqualToUsermode,
    kMPU_UserNoAccessRights,
    kMPU_IdentifierDisable,
    kMPU_SupervisorEqualToUsermode,
    kMPU_UserNoAccessRights,
    kMPU_IdentifierDisable,
    kMPU_SupervisorEqualToUsermode,
    kMPU_UserNoAccessRights,
    kMPU_IdentifierDisable
};

mpu_rwrights_master_access_control_t mpuRwAccessRightsMasters =
{
    false,
    false,
    false,
    false,
    false,
    false,
    false,
    false,
    false
};

// Defines the MPU region configuration structure. //
mpu_region_config_t mpuRegionConfig =
{
    0,
    0x0,
    0xffffffff,
    mpuRwxAccessRightsMasters,
    mpuRwAccessRightsMasters,
```

## Basic Control Operations

```
    0,  
    0  
};  
  
// Defines the MPU user configuration structure.  
mpu_config_t mpuUserConfig =  
{  
    mpuRegionConfig,  
    NULL  
};  
  
// Initializes the MPU region 0.  
MPU_Init(MPU, &mpuUserConfig);
```

### 23.3 Basic Control Operations

MPU can be enabled/disabled for the entire memory protection region by calling the [MPU\\_Enable\(\)](#). To save the power for any unused special regions when the entire memory protection region is disabled, call the [MPU\\_RegionEnable\(\)](#).

After MPU initialization, the [MPU\\_SetRegionLowMasterAccessRights\(\)](#) and [MPU\\_SetRegionHighMasterAccessRights\(\)](#) can be used to change the access rights for special master ports and for special region numbers. The [MPU\\_SetRegionConfig](#) can be used to set the whole region with the start/end address with access rights.

The [MPU\\_GetHardwareInfo\(\)](#) API is provided to get the hardware information for the device. The [MPU\\_GetSlavePortErrorStatus\(\)](#) API is provided to get the error status of a special slave port. When an error happens in this port, the [MPU\\_GetDetailErrorAccessInfo\(\)](#) API is provided to get the detailed error information.

## Data Structures

- struct [mpu\\_hardware\\_info\\_t](#)  
*MPU hardware basic information.* [More...](#)
- struct [mpu\\_access\\_err\\_info\\_t](#)  
*MPU detail error access information.* [More...](#)
- struct [mpu\\_rwxrights\\_master\\_access\\_control\\_t](#)  
*MPU read/write/execute rights control for bus master 0 ~ 3.* [More...](#)
- struct [mpu\\_rwrights\\_master\\_access\\_control\\_t](#)  
*MPU read/write access control for bus master 4 ~ 7.* [More...](#)
- struct [mpu\\_region\\_config\\_t](#)  
*MPU region configuration structure.* [More...](#)
- struct [mpu\\_config\\_t](#)  
*The configuration structure for the MPU initialization.* [More...](#)

## Macros

- #define [MPU\\_REGION\\_RWXRIGHTS\\_MASTER\\_SHIFT\(n\)](#) ( $n * 6$ )  
*MPU the bit shift for masters with privilege rights: read write and execute.*
- #define [MPU\\_REGION\\_RWXRIGHTS\\_MASTER\\_MASK\(n\)](#) ( $0x1Fu << \text{MPU\_REGION\_RWXRIGHTS\_MASTER\_SHIFT}(n)$ )  
*MPU masters with read, write and execute rights bit mask.*
- #define [MPU\\_REGION\\_RWXRIGHTS\\_MASTER\\_WIDTH](#) 5

- *MPU masters with read, write and execute rights bit width.*  
 • #define **MPU\_REGION\_RWXRIGHTS\_MASTER(n, x)** (((uint32\_t)((uint32\_t)(x)) << **MPU\_REGION\_RWXRIGHTS\_MASTER\_SHIFT(n)**) & **MPU\_REGION\_RWXRIGHTS\_MASTER\_MASK(n)**)
  - MPU masters with read, write and execute rights priority setting.*
- #define **MPU\_REGION\_RWXRIGHTS\_MASTER\_PE\_SHIFT(n)** (n \* 6 + **MPU\_REGION\_RWXRIGHTS\_MASTER\_WIDTH**)
  - MPU masters with read, write and execute rights process enable bit shift.*
- #define **MPU\_REGION\_RWXRIGHTS\_MASTER\_PE\_MASK(n)** (0x1u << **MPU\_REGION\_RWXRIGHTS\_MASTER\_PE\_SHIFT(n)**)
  - MPU masters with read, write and execute rights process enable bit mask.*
- #define **MPU\_REGION\_RWXRIGHTS\_MASTER\_PE(n, x)** (((uint32\_t)((uint32\_t)(x)) << **MPU\_REGION\_RWXRIGHTS\_MASTER\_PE\_SHIFT(n)**) & **MPU\_REGION\_RWXRIGHTS\_MASTER\_PE\_MASK(n)**)
  - MPU masters with read, write and execute rights process enable setting.*
- #define **MPU\_REGION\_RWRIGHTS\_MASTER\_SHIFT(n)** ((n - FSL\_FEATURE\_MPU\_PRIVILEGED\_RIGHTS\_MASTER\_COUNT) \* 2 + 24)
  - MPU masters with normal read write permission bit shift.*
- #define **MPU\_REGION\_RWRIGHTS\_MASTER\_MASK(n)** (0x3u << **MPU\_REGION\_RWRIGHTS\_MASTER\_SHIFT(n)**)
  - MPU masters with normal read write rights bit mask.*
- #define **MPU\_REGION\_RWRIGHTS\_MASTER(n, x)** (((uint32\_t)((uint32\_t)(x)) << **MPU\_REGION\_RWRIGHTS\_MASTER\_SHIFT(n)**) & **MPU\_REGION\_RWRIGHTS\_MASTER\_MASK(n)**)
  - MPU masters with normal read write rights priority setting.*

## Enumerations

- enum **mpu\_region\_total\_num\_t** {
   
kMPU\_8Regions = 0x0U,  
kMPU\_12Regions = 0x1U,  
kMPU\_16Regions = 0x2U }
  - Describes the number of MPU regions.*
- enum **mpu\_slave\_t** {
   
kMPU\_Slave0 = 4U,  
kMPU\_Slave1 = 3U,  
kMPU\_Slave2 = 2U,  
kMPU\_Slave3 = 1U,  
kMPU\_Slave4 = 0U }
  - MPU slave port number.*
- enum **mpu\_err\_access\_control\_t** {
   
kMPU\_NoRegionHit = 0U,  
kMPU\_NoneOverlappRegion = 1U,  
kMPU\_OverlappRegion = 2U }
  - MPU error access control detail.*
- enum **mpu\_err\_access\_type\_t** {
   
kMPU\_ErrTypeRead = 0U,  
kMPU\_ErrTypeWrite = 1U }

## Basic Control Operations

- *MPU error access type.*  
• enum `mpu_err_attributes_t` {  
  `kMPU_InstructionAccessInUserMode` = 0U,  
  `kMPU_DataAccessInUserMode` = 1U,  
  `kMPU_InstructionAccessInSupervisorMode` = 2U,  
  `kMPU_DataAccessInSupervisorMode` = 3U }  
*MPU access error attributes.*
- enum `mpu_supervisor_access_rights_t` {  
  `kMPU_SupervisorReadWriteExecute` = 0U,  
  `kMPU_SupervisorReadExecute` = 1U,  
  `kMPU_SupervisorReadWrite` = 2U,  
  `kMPU_SupervisorEqualToUsermode` = 3U }  
*MPU access rights in supervisor mode for bus master 0 ~ 3.*
- enum `mpu_user_access_rights_t` {  
  `kMPU_UserNoAccessRights` = 0U,  
  `kMPU_UserExecute` = 1U,  
  `kMPU_UserWrite` = 2U,  
  `kMPU_UserWriteExecute` = 3U,  
  `kMPU_UserRead` = 4U,  
  `kMPU_UserReadExecute` = 5U,  
  `kMPU_UserReadWrite` = 6U,  
  `kMPU_UserReadWriteExecute` = 7U }  
*MPU access rights in user mode for bus master 0 ~ 3.*

## Driver version

- #define `FSL_MPUM_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 0)`)  
*MPU driver version 2.1.0.*

## Initialization and deinitialization

- void `MPU_Init` (`MPU_Type` \*base, const `mpu_config_t` \*config)  
*Initializes the MPU with the user configuration structure.*
- void `MPU_Deinit` (`MPU_Type` \*base)  
*Deinitializes the MPU regions.*

## Basic Control Operations

- static void `MPU_Enable` (`MPU_Type` \*base, bool enable)  
*Enables/disables the MPU globally.*
- static void `MPU_RegionEnable` (`MPU_Type` \*base, `uint32_t` number, bool enable)  
*Enables/disables the MPU for a special region.*
- void `MPU_GetHardwareInfo` (`MPU_Type` \*base, `mpu_hardware_info_t` \*hardwareInform)  
*Gets the MPU basic hardware information.*
- void `MPU_SetRegionConfig` (`MPU_Type` \*base, const `mpu_region_config_t` \*regionConfig)  
*Sets the MPU region.*
- void `MPU_SetRegionAddr` (`MPU_Type` \*base, `uint32_t` regionNum, `uint32_t` startAddr, `uint32_t` endAddr)

- `void MPU_SetRegionRwxMasterAccessRights` (`MPU_Type *base, uint32_t regionNum, uint32_t masterNum, const mpu_rwxrights_master_access_control_t *accessRights`)
 

*Sets the MPU region start and end address.*
- `void MPU_SetRegionRwMasterAccessRights` (`MPU_Type *base, uint32_t regionNum, uint32_t masterNum, const mpu_rwrights_master_access_control_t *accessRights`)
 

*Sets the MPU region access rights for masters with read, write and execute rights.*
- `bool MPU_GetSlavePortErrorStatus` (`MPU_Type *base, mpu_slave_t slaveNum`)
 

*Gets the numbers of slave ports where errors occur.*
- `void MPU_GetDetailErrorAccessInfo` (`MPU_Type *base, mpu_slave_t slaveNum, mpu_access_err_info_t *errInform`)
 

*Gets the MPU detailed error access information.*

## 23.4 Data Structure Documentation

### 23.4.1 struct mpu\_hardware\_info\_t

#### Data Fields

- `uint8_t hardwareRevisionLevel`

*Specifies the MPU's hardware and definition reversion level.*
- `uint8_t slavePortsNumbers`

*Specifies the number of slave ports connected to MPU.*
- `mpu_region_total_num_t regionsNumbers`

*Indicates the number of region descriptors implemented.*

#### 23.4.1.0.5.1 Field Documentation

##### 23.4.1.0.5.1.1 uint8\_t mpu\_hardware\_info\_t::hardwareRevisionLevel

##### 23.4.1.0.5.1.2 uint8\_t mpu\_hardware\_info\_t::slavePortsNumbers

##### 23.4.1.0.5.1.3 mpu\_region\_total\_num\_t mpu\_hardware\_info\_t::regionsNumbers

### 23.4.2 struct mpu\_access\_err\_info\_t

#### Data Fields

- `uint32_t master`

*Access error master.*
- `mpu_err_attributes_t attributes`

*Access error attributes.*
- `mpu_err_access_type_t accessType`

*Access error type.*
- `mpu_err_access_control_t accessControl`

*Access error control.*
- `uint32_t address`

*Access error address.*

## Data Structure Documentation

### 23.4.2.0.5.2 Field Documentation

23.4.2.0.5.2.1 `uint32_t mpu_access_err_info_t::master`

23.4.2.0.5.2.2 `mpu_err_attributes_t mpu_access_err_info_t::attributes`

23.4.2.0.5.2.3 `mpu_err_access_type_t mpu_access_err_info_t::accessType`

23.4.2.0.5.2.4 `mpu_err_access_control_t mpu_access_err_info_t::accessControl`

23.4.2.0.5.2.5 `uint32_t mpu_access_err_info_t::address`

### 23.4.3 `struct mpu_rwxrights_master_access_control_t`

#### Data Fields

- `mpu_supervisor_access_rights_t superAccessRights`  
*Master access rights in supervisor mode.*
- `mpu_user_access_rights_t userAccessRights`  
*Master access rights in user mode.*

### 23.4.3.0.5.3 Field Documentation

23.4.3.0.5.3.1 `mpu_supervisor_access_rights_t mpu_rwxrights_master_access_control_t::superAccessRights`

23.4.3.0.5.3.2 `mpu_user_access_rights_t mpu_rwxrights_master_access_control_t::userAccessRights`

### 23.4.4 `struct mpu_rwrights_master_access_control_t`

#### Data Fields

- `bool writeEnable`  
*Enables or disables write permission.*
- `bool readEnable`  
*Enables or disables read permission.*

### 23.4.4.0.5.4 Field Documentation

23.4.4.0.5.4.1 `bool mpu_rwrights_master_access_control_t::writeEnable`

23.4.4.0.5.4.2 `bool mpu_rwrights_master_access_control_t::readEnable`

### 23.4.5 `struct mpu_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] ~ access-

Rights2[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: MPU assigns a priority scheme where the debugger is treated as the highest priority master followed by the core and then all the remaining masters. MPU 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 guarantee 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`  
*MPU region number, range form 0 ~ FSL\_FEATURE\_MPU\_DESCRIPTOR\_COUNT - 1.*
- `uint32_t startAddress`  
*Memory region start address.*
- `uint32_t endAddress`  
*Memory region end address.*
- `mpu_rwxrights_master_access_control_t accessRights1 [4]`  
*Masters with read, write and execute rights setting.*
- `mpu_rwrights_master_access_control_t accessRights2 [4]`  
*Masters with normal read write rights setting.*

### 23.4.5.0.5.5 Field Documentation

#### 23.4.5.0.5.5.1 `uint32_t mpu_region_config_t::regionNum`

#### 23.4.5.0.5.5.2 `uint32_t mpu_region_config_t::startAddress`

Note: bit0 ~ bit4 always be marked as 0 by MPU. The actual start address is 0-modulo-32 byte address.

#### 23.4.5.0.5.5.3 `uint32_t mpu_region_config_t::endAddress`

Note: bit0 ~ bit4 always be marked as 1 by MPU. The actual end address is 31-modulo-32 byte address.

#### 23.4.5.0.5.5.4 `mpu_rwxrights_master_access_control_t mpu_region_config_t::accessRights1[4]`

#### 23.4.5.0.5.5.5 `mpu_rwrights_master_access_control_t mpu_region_config_t::accessRights2[4]`

## 23.4.6 `struct mpu_config_t`

This structure is used when calling the MPU\_Init function.

## Data Fields

- `mpu_region_config_t regionConfig`  
*region access permission.*
- `struct _mpu_config * next`

## Data Structure Documentation

*pointer to the next structure.*

### 23.4.6.0.5.6 Field Documentation

23.4.6.0.5.6.1 `mpu_region_config_t mpu_config_t::regionConfig`

23.4.6.0.5.6.2 `struct _mpu_config* mpu_config_t::next`

## 23.5 Macro Definition Documentation

- 23.5.1 `#define FSL_MPU_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`
- 23.5.2 `#define MPU_REGION_RWXRIGHTS_MASTER_SHIFT( n ) (n * 6)`
- 23.5.3 `#define MPU_REGION_RWXRIGHTS_MASTER_MASK( n ) (0x1Fu << MPU_REGION_RWXRIGHTS_MASTER_SHIFT(n))`
- 23.5.4 `#define MPU_REGION_RWXRIGHTS_MASTER_WIDTH 5`
- 23.5.5 `#define MPU_REGION_RWXRIGHTS_MASTER( n, x ) (((uint32_t)((uint32_t)(x)) << MPU_REGION_RWXRIGHTS_MASTER_SHIFT(n))) & MPU_REGION_RWXRIGHTS_MASTER_MASK(n))`
- 23.5.6 `#define MPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT( n ) (n * 6 + MPU_REGION_RWXRIGHTS_MASTER_WIDTH)`
- 23.5.7 `#define MPU_REGION_RWXRIGHTS_MASTER_PE_MASK( n ) (0x1u << MPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT(n))`
- 23.5.8 `#define MPU_REGION_RWXRIGHTS_MASTER_PE( n, x ) (((uint32_t)((uint32_t)(x)) << MPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT(n))) & MPU_REGION_RWXRIGHTS_MASTER_PE_MASK(n))`
- 23.5.9 `#define MPU_REGION_RWRIGHTS_MASTER_SHIFT( n ) ((n - FSL_FEATURE_MPU_PRIVILEGED_RIGHTS_MASTER_COUNT) * 2 + 24)`
- 23.5.10 `#define MPU_REGION_RWRIGHTS_MASTER_MASK( n ) (0x3u << MPU_REGION_RWRIGHTS_MASTER_SHIFT(n))`
- 23.5.11 `#define MPU_REGION_RWRIGHTS_MASTER( n, x ) (((uint32_t)((uint32_t)(x)) << MPU_REGION_RWRIGHTS_MASTER_SHIFT(n))) & MPU_REGION_RWRIGHTS_MASTER_MASK(n))`

## Enumeration Type Documentation

### 23.6 Enumeration Type Documentation

#### 23.6.1 enum mpu\_region\_total\_num\_t

Enumerator

*kMPU\_8Regions* MPU supports 8 regions.

*kMPU\_12Regions* MPU supports 12 regions.

*kMPU\_16Regions* MPU supports 16 regions.

#### 23.6.2 enum mpu\_slave\_t

Enumerator

*kMPU\_Slave0* MPU slave port 0.

*kMPU\_Slave1* MPU slave port 1.

*kMPU\_Slave2* MPU slave port 2.

*kMPU\_Slave3* MPU slave port 3.

*kMPU\_Slave4* MPU slave port 4.

#### 23.6.3 enum mpu\_err\_access\_control\_t

Enumerator

*kMPU\_NoRegionHit* No region hit error.

*kMPU\_NoneOverlapRegion* Access single region error.

*kMPU\_OverlapRegion* Access overlapping region error.

#### 23.6.4 enum mpu\_err\_access\_type\_t

Enumerator

*kMPU\_ErrTypeRead* MPU error access type — read.

*kMPU\_ErrTypeWrite* MPU error access type — write.

#### 23.6.5 enum mpu\_err\_attributes\_t

Enumerator

*kMPU\_InstructionAccessInUserMode* Access instruction error in user mode.

*kMPU\_DataAccessInUserMode* Access data error in user mode.

*kMPU\_InstructionAccessInSupervisorMode* Access instruction error in supervisor mode.

*kMPU\_DataAccessInSupervisorMode* Access data error in supervisor mode.

### 23.6.6 enum mpu\_supervisor\_access\_rights\_t

Enumerator

- kMPU\_SupervisorReadWriteExecute*** Read write and execute operations are allowed in supervisor mode.
- kMPU\_SupervisorReadExecute*** Read and execute operations are allowed in supervisor mode.
- kMPU\_SupervisorReadWrite*** Read write operations are allowed in supervisor mode.
- kMPU\_SupervisorEqualToUsermode*** Access permission equal to user mode.

### 23.6.7 enum mpu\_user\_access\_rights\_t

Enumerator

- kMPU\_UserNoAccessRights*** No access allowed in user mode.
- kMPU\_UserExecute*** Execute operation is allowed in user mode.
- kMPU\_UserWrite*** Write operation is allowed in user mode.
- kMPU\_UserWriteExecute*** Write and execute operations are allowed in user mode.
- kMPU\_UserRead*** Read is allowed in user mode.
- kMPU\_UserReadExecute*** Read and execute operations are allowed in user mode.
- kMPU\_UserReadWrite*** Read and write operations are allowed in user mode.
- kMPU\_UserReadWriteExecute*** Read write and execute operations are allowed in user mode.

## 23.7 Function Documentation

### 23.7.1 void MPU\_Init ( MPU\_Type \* *base*, const mpu\_config\_t \* *config* )

This function configures the MPU module with the user-defined configuration.

Parameters

<i>base</i>	MPU peripheral base address.
<i>config</i>	The pointer to the configuration structure.

### 23.7.2 void MPU\_Deinit ( MPU\_Type \* *base* )

Parameters

## Function Documentation

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

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

Call this API to enable or disable the MPU module.

Parameters

<i>base</i>	MPU peripheral base address.
<i>enable</i>	True enable MPU, false disable MPU.

### 23.7.4 static void MPU\_RegionEnable ( MPU\_Type \* *base*, uint32\_t *number*, bool *enable* ) [inline], [static]

When MPU is enabled, call this API to disable an unused region of an enabled MPU. Call this API to minimize the power dissipation.

Parameters

<i>base</i>	MPU peripheral base address.
<i>number</i>	MPU region number.
<i>enable</i>	True enable the special region MPU, false disable the special region MPU.

### 23.7.5 void MPU\_GetHardwareInfo ( MPU\_Type \* *base*, mpu.hardware\_info\_t \* *hardwareInform* )

Parameters

<i>base</i>	MPU peripheral base address.
<i>hardware-Inform</i>	The pointer to the MPU hardware information structure. See "mpu.hardware_info_t".

### 23.7.6 void MPU\_SetRegionConfig ( MPU\_Type \* *base*, const mpu\_region\_config\_t \* *regionConfig* )

Note: Due to the MPU 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>	MPU peripheral base address.
<i>regionConfig</i>	The pointer to the MPU user configuration structure. See "mpu_region_config_t".

### 23.7.7 void MPU\_SetRegionAddr ( MPU\_Type \* *base*, uint32\_t *regionNum*, uint32\_t *startAddr*, uint32\_t *endAddr* )

Memory region start address. Note: bit0 ~ bit4 is always marked as 0 by MPU. The actual start address by MPU is 0-modulo-32 byte address. Memory region end address. Note: bit0 ~ bit4 always be marked as 1 by MPU. The actual end address used by MPU is 31-modulo-32 byte address. Note: Due to the MPU protection, the startAddr and endAddr can't be changed by the core when regionNum is 0.

Parameters

<i>base</i>	MPU peripheral base address.
<i>regionNum</i>	MPU region number. The range is from 0 to FSL_FEATURE_MPU_DESCRIPTOR_COUNT - 1.
<i>startAddr</i>	Region start address.
<i>endAddr</i>	Region end address.

### 23.7.8 void MPU\_SetRegionRwxMasterAccessRights ( MPU\_Type \* *base*, uint32\_t *regionNum*, uint32\_t *masterNum*, const mpu\_rwxrights\_master\_access\_control\_t \* *accessRights* )

The MPU 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. So except the normal read and write rights, the execute rights is also allowed for these masters. The privilege rights masters are normally range from bus masters 0 - 3. However, the maximum master number is device-specific. See the "FSL\_FEATURE\_MPU\_PRIVILEGED\_RIGHTS\_MASTER\_MAX\_INDEX". The normal rights masters access rights control see "MPU\_SetRegionRwMasterAccessRights()".

## Function Documentation

Parameters

<i>base</i>	MPU peripheral base address.
<i>regionNum</i>	MPU region number. Should range from 0 to FSL FEATURE MPU_DESCRIPTOR_COUNT - 1.
<i>masterNum</i>	MPU bus master number. Should range from 0 to FSL FEATURE MPU_PRIVILEGED_RIGHTS_MASTER_MAX_INDEX.
<i>accessRights</i>	The pointer to the MPU access rights configuration. See "mpu_rwxrights_master_access_control_t".

**23.7.9 void MPU\_SetRegionRwMasterAccessRights ( MPU\_Type \* *base*, uint32\_t *regionNum*, uint32\_t *masterNum*, const mpu\_rwrights\_master\_access\_control\_t \* *accessRights* )**

The MPU access rights depend on two board classifications of bus masters. The privilege rights masters and the normal rights masters. The normal rights masters only have the read and write access permissions. The privilege rights access control see "MPU\_SetRegionRwxMasterAccessRights".

Parameters

<i>base</i>	MPU peripheral base address.
<i>regionNum</i>	MPU region number. The range is from 0 to FSL FEATURE MPU_DESCRIPTOR_COUNT - 1.
<i>masterNum</i>	MPU bus master number. Should range from FSL FEATURE MPU_PRIVILEGED_RIGHTS_MASTER_COUNT to ~ FSL FEATURE MPU_MASTER_MAX_INDEX.
<i>accessRights</i>	The pointer to the MPU access rights configuration. See "mpu_rwrights_master_access_control_t".

**23.7.10 bool MPU\_GetSlavePortErrorStatus ( MPU\_Type \* *base*, mpu\_slave\_t *slaveNum* )**

Parameters

<i>base</i>	MPU peripheral base address.
<i>slaveNum</i>	MPU 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.

### 23.7.11 void MPU\_GetDetailErrorAccessInfo ( **MPU\_Type** \* *base*, **mpu\_slave\_t** *slaveNum*, **mpu\_access\_err\_info\_t** \* *errInform* )

Parameters

<i>base</i>	MPU peripheral base address.
<i>slaveNum</i>	MPU slave port number.
<i>errInform</i>	The pointer to the MPU access error information. See "mpu_access_err_info_t".

## Function Documentation

# Chapter 24

## PIT: Periodic Interrupt Timer

### 24.1 Overview

The KSDK provides a driver for the Periodic Interrupt Timer (PIT) of Kinetis devices.

### 24.2 Function groups

The PIT driver supports operating the module as a time counter.

#### 24.2.1 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.

#### 24.2.2 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. User can call the utility macros provided in `fsl_common.h` to convert to microseconds or milliseconds

#### 24.2.3 Start and Stop timer operations

The function [PIT\\_StartTimer\(\)](#) starts the timer counting. After calling this function, the timer loads the period value set earlier via the [PIT\\_SetPeriod\(\)](#) function and starts counting down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

The function [PIT\\_StopTimer\(\)](#) stops the timer counting.

## Typical use case

### 24.2.4 Status

Provides functions to get and clear the PIT status.

### 24.2.5 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

## 24.3 Typical use case

### 24.3.1 PIT tick example

Updates the PIT period and toggles an LED periodically.

```
int main(void)
{
    /* Structure of initialize PIT */
    pit_config_t pitConfig;

    /* Initialize and enable LED */
    LED_INIT();

    /* Board pin, clock, debug console init */
    BOARD_InitHardware();

    PIT_GetDefaultConfig(&pitConfig);

    /* Init pit module */
    PIT_Init(PIT, &pitConfig);

    /* Set timer period for channel 0 */
    PIT_SetTimerPeriod(PIT, kPIT_Chnl_0, USEC_TO_COUNT(1000000U,
        PIT_SOURCE_CLOCK));

    /* Enable timer interrupts for channel 0 */
    PIT_EnableInterrupts(PIT, kPIT_Chnl_0,
        kPIT_TimerInterruptEnable);

    /* Enable at the NVIC */
    EnableIRQ(PIT IRQ_ID);

    /* Start channel 0 */
    PRINTF("\r\nStarting channel No.0 ...");
    PIT_StartTimer(PIT, kPIT_Chnl_0);

    while (true)
    {
        /* Check whether occur interrupt and toggle LED */
        if (true == pitIsrFlag)
        {
            PRINTF("\r\n Channel No.0 interrupt is occurred !");
            LED_TOGGLE();
            pitIsrFlag = false;
        }
    }
}
```

## Data Structures

- struct [pit\\_config\\_t](#)  
*PIT config 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 }
  - enum [pit\\_status\\_flags\\_t](#) { kPIT\_TimerFlag = PIT\_TFLG\_TIF\_MASK }
- List of PIT interrupts.*
- List of PIT status flags.*

## Driver version

- #define [FSL\\_PIT\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 0))
   
*Version 2.0.0.*

## 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 operation.*
- void [PIT\\_Deinit](#) (PIT\_Type \*base)
   
*Gates the PIT clock and disable the PIT module.*
- static void [PIT\\_GetDefaultConfig](#) ([pit\\_config\\_t](#) \*config)
   
*Fill in the PIT config struct with the default settings.*

## Interrupt Interface

- static void [PIT\\_EnableInterrupts](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel, uint32\_t mask)
   
*Enables the selected PIT interrupts.*
- static void [PIT\\_DisableInterrupts](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel, uint32\_t mask)
   
*Disables the selected PIT interrupts.*
- static uint32\_t [PIT\\_GetEnabledInterrupts](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel)
   
*Gets the enabled PIT interrupts.*

## Status Interface

- static uint32\_t [PIT\\_GetStatusFlags](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel)
   
*Gets the PIT status flags.*
- static void [PIT\\_ClearStatusFlags](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel, uint32\_t mask)
   
*Clears the PIT status flags.*

## Enumeration Type Documentation

### 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.*

## 24.4 Data Structure Documentation

### 24.4.1 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 config struct can be made const so it resides in flash

### Data Fields

- bool [enableRunInDebug](#)  
*true: Timers run in debug mode; false: Timers stop in debug mode*

## 24.5 Enumeration Type Documentation

### 24.5.1 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.

### 24.5.2 enum pit\_interrupt\_enable\_t

Enumerator

*kPIT\_TimerInterruptEnable* Timer interrupt enable.

### 24.5.3 enum pit\_status\_flags\_t

Enumerator

*kPIT\_TimerFlag* Timer flag.

## 24.6 Function Documentation

### 24.6.1 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 user's PIT config structure

### 24.6.2 void PIT\_Deinit ( PIT\_Type \* *base* )

Parameters

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

### 24.6.3 static void PIT\_GetDefaultConfig ( pit\_config\_t \* *config* ) [inline], [static]

The default values are:

```
*     config->enableRunInDebug = false;
*
```

## Function Documentation

Parameters

<i>config</i>	Pointer to user's PIT config structure.
---------------	---

### 24.6.4 static void PIT\_EnableInterrupts ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel*, **uint32\_t** *mask* ) [inline], [static]

Parameters

<i>base</i>	PIT peripheral base address
<i>channel</i>	Timer channel number
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">pit_interrupt_enable_t</a>

### 24.6.5 static void PIT\_DisableInterrupts ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel*, **uint32\_t** *mask* ) [inline], [static]

Parameters

<i>base</i>	PIT peripheral base address
<i>channel</i>	Timer channel number
<i>mask</i>	The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">pit_interrupt_enable_t</a>

### 24.6.6 static uint32\_t PIT\_GetEnabledInterrupts ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel* ) [inline], [static]

Parameters

<i>base</i>	PIT peripheral base address
<i>channel</i>	Timer channel number

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [pit\\_interrupt\\_enable\\_t](#)

24.6.7 **static uint32\_t PIT\_GetStatusFlags ( PIT\_Type \* *base*, pit\_chnl\_t *channel* )**  
[**inline**], [**static**]

## Function Documentation

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](#)

### 24.6.8 static void PIT\_ClearStatusFlags ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel*, **uint32\_t** *mask* ) [inline], [static]

Parameters

<i>base</i>	PIT peripheral base address
<i>channel</i>	Timer channel number
<i>mask</i>	The status flags to clear. This is a logical OR of members of the enumeration <a href="#">pit_status_flags_t</a>

### 24.6.9 static void PIT\_SetTimerPeriod ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel*, **uint32\_t** *count* ) [inline], [static]

Timers begin counting from the value set by this function until it reaches 0, then it generates an interrupt and load this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

Note

User 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
--------------	--------------------------------

#### 24.6.10 static uint32\_t PIT\_GetCurrentTimerCount ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel* ) [inline], [static]

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note

User 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

#### 24.6.11 static void PIT\_StartTimer ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel* ) [inline], [static]

After calling this function, timers load period value, count down to 0 and then load the respective start value again. Each time a timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

<i>base</i>	PIT peripheral base address
<i>channel</i>	Timer channel number.

#### 24.6.12 static void PIT\_StopTimer ( **PIT\_Type** \* *base*, **pit\_chnl\_t** *channel* ) [inline], [static]

This function stops every timer counting. Timers reload their periods respectively after the next time they call the PIT\_DRV\_StartTimer.

## Function Documentation

### Parameters

<i>base</i>	PIT peripheral base address
<i>channel</i>	Timer channel number.

# Chapter 25

## PMC: Power Management Controller

### 25.1 Overview

The KSDK provides a Peripheral driver for the Power Management Controller (PMC) module of Kinetis 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...](#)*

### Driver version

- #define [FSL\\_PMC\\_DRIVER\\_VERSION\(MAKE\\_VERSION\(2, 0, 0\)\)](#)  
*PMC driver version.*

### Power Management Controller Control APIs

- void [PMC\\_ConfigureLowVoltDetect](#) (PMC\_Type \*base, const [pmc\\_low\\_volt\\_detect\\_config\\_t](#) \*config)  
*Configure the low-voltage detect setting.*
- static bool [PMC\\_GetLowVoltDetectFlag](#) (PMC\_Type \*base)  
*Get Low-Voltage Detect Flag status.*
- static void [PMC\\_ClearLowVoltDetectFlag](#) (PMC\_Type \*base)  
*Acknowledge to clear the Low-voltage Detect flag.*
- void [PMC\\_ConfigureLowVoltWarning](#) (PMC\_Type \*base, const [pmc\\_low\\_volt\\_warning\\_config\\_t](#) \*config)  
*Configure the low-voltage warning setting.*
- static bool [PMC\\_GetLowVoltWarningFlag](#) (PMC\_Type \*base)  
*Get Low-Voltage Warning Flag status.*
- static void [PMC\\_ClearLowVoltWarningFlag](#) (PMC\_Type \*base)  
*Acknowledge to Low-Voltage Warning flag.*

### 25.2 Data Structure Documentation

#### 25.2.1 struct pmc\_low\_volt\_detect\_config\_t

### Data Fields

- bool [enableInt](#)

## Function Documentation

- `bool enableReset`  
*Enable system reset when low-voltage detect.*

### 25.2.2 `struct pmc_low_volt_warning_config_t`

#### Data Fields

- `bool enableInt`  
*Enable interrupt when low-voltage warning.*

## 25.3 Macro Definition Documentation

### 25.3.1 `#define FSL_PMC_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

Version 2.0.0.

## 25.4 Function Documentation

### 25.4.1 `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, enable interrupt or not, enable system reset or not.

Parameters

<code>base</code>	PMC peripheral base address.
<code>config</code>	Low-Voltage detect configuration structure.

### 25.4.2 `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

<code>base</code>	PMC peripheral base address.
-------------------	------------------------------

Returns

Current low-voltage detect flag

- true: Low-voltage detected
- false: Low-voltage not detected

#### **25.4.3 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.
-------------	------------------------------

#### **25.4.4 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 enable interrupt or not.

Parameters

<i>base</i>	PMC peripheral base address.
<i>config</i>	Low-Voltage warning configuration structure.

#### **25.4.5 static bool PMC\_GetLowVoltWarningFlag ( **PMC\_Type** \* *base* ) [inline], [static]**

This function polls the current LWWF status. When 1 is returned, it indicates a low-voltage warning event. LWWF 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 LWWF status

- true: Low-Voltage Warning Flag is set.
- false: the Low-Voltage Warning does not happen.

## Function Documentation

**25.4.6 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.
-------------	------------------------------

## Function Documentation

# Chapter 26

## PORT: Port Control and Interrupts

### 26.1 Overview

The KSDK provides a driver for the Port Control and Interrupts (PORT) module of Kinetis devices.

### 26.2 Typical configuration use case

#### 26.2.1 Input PORT configuration

```
/* Input pin PORT configuration */
port_pin_config_t config = {
    kPORT_PullUp,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnLockRegister,
};

/* Sets the configuration */
PORT_SetPinConfig(PORTA, 4, &config);
```

#### 26.2.2 I2C PORT Configuration

```
/* I2C pin PORTconfiguration */
port_pin_config_t config = {
    kPORT_PullUp,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainEnable,
    kPORT_LowDriveStrength,
    kPORT_MuxAlt5,
    kPORT_UnLockRegister,
};

PORT_SetPinConfig(PORTE, 24u, &config);
PORT_SetPinConfig(PORTE, 25u, &config);
```

## Data Structures

- struct `port_pin_config_t`  
*PORT pin configuration structure.* [More...](#)

## Enumerations

- enum `_port_pull` {  
    kPORT\_PullDisable = 0U,  
    kPORT\_PullDown = 2U,  
    kPORT\_PullUp = 3U }

## Typical configuration use case

- Internal resistor pull feature selection.
  - enum `_port_slew_rate` {  
    `kPORT_FastSlewRate` = 0U,  
    `kPORT_SlowSlewRate` = 1U }
- Slew rate selection.
  - enum `_port_passive_filter_enable` {  
    `kPORT_PassiveFilterDisable` = 0U,  
    `kPORT_PassiveFilterEnable` = 1U }
- Passive filter feature enable/disable.
  - enum `_port_drive_strength` {  
    `kPORT_LowDriveStrength` = 0U,  
    `kPORT_HighDriveStrength` = 1U }
- Configures the drive strength.
  - enum `port_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 }
- Pin mux selection.
  - enum `port_interrupt_t` {  
    `kPORT_InterruptOrDMADisabled` = 0x0U,  
    `kPORT_InterruptLogicZero` = 0x8U,  
    `kPORT_InterruptRisingEdge` = 0x9U,  
    `kPORT_InterruptFallingEdge` = 0xAU,  
    `kPORT_InterruptEitherEdge` = 0xBU,  
    `kPORT_InterruptLogicOne` = 0xCU }
- Configures the interrupt generation condition.

## Driver version

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

## Configuration

- static void `PORT_SetPinConfig` (`PORT_Type` \*base, `uint32_t` pin, const `port_pin_config_t` \*config)  
*Sets the port PCR register.*
- static void `PORT_SetMultiplePinsConfig` (`PORT_Type` \*base, `uint32_t` mask, const `port_pin_config_t` \*config)  
*Sets the port PCR register for multiple pins.*
- static void `PORT_SetPinMux` (`PORT_Type` \*base, `uint32_t` pin, `port_mux_t` mux)  
*Configures the pin muxing.*

## 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.*

## 26.3 Data Structure Documentation

### 26.3.1 struct port\_pin\_config\_t

#### Data Fields

- uint16\_t **pullSelect**: 2  
*No-pull/pull-down/pull-up select.*
- uint16\_t **slewRate**: 1  
*Fast/slow slew rate Configure.*
- uint16\_t **passiveFilterEnable**: 1  
*Passive filter enable/disable.*
- uint16\_t **driveStrength**: 1  
*Fast/slow drive strength configure.*
- uint16\_t **mux**: 3  
*Pin mux Configure.*

## 26.4 Macro Definition Documentation

### 26.4.1 #define FSL\_PORT\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

## 26.5 Enumeration Type Documentation

### 26.5.1 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.

### 26.5.2 enum \_port\_slew\_rate

Enumerator

- kPORT\_FastSlewRate** Fast slew rate is configured.  
**kPORT\_SlowSlewRate** Slow slew rate is configured.

## Function Documentation

### 26.5.3 enum \_port\_passive\_filter\_enable

Enumerator

*kPORT\_PassiveFilterDisable* Fast slew rate is configured.

*kPORT\_PassiveFilterEnable* Slow slew rate is configured.

### 26.5.4 enum \_port\_drive\_strength

Enumerator

*kPORT\_LowDriveStrength* Low-drive strength is configured.

*kPORT\_HighDriveStrength* High-drive strength is configured.

### 26.5.5 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.

### 26.5.6 enum port\_interrupt\_t

Enumerator

*kPORT\_InterruptOrDMA.Disabled* Interrupt/DMA request is disabled.

*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.

## 26.6 Function Documentation

### 26.6.1 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.

### 26.6.2 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.

### 26.6.3 static void PORT\_SetPinMux ( PORT\_Type \* *base*, uint32\_t *pin*, port\_mux\_t *mux* ) [inline], [static]

## Function Documentation

### Parameters

<i>base</i>	PORT peripheral base pointer.
<i>pin</i>	PORT pin number.
<i>mux</i>	pin muxing slot selection. <ul style="list-style-type: none"><li>• <a href="#">kPORT_PinDisabledOrAnalog</a>: Pin disabled or work in analog function.</li><li>• <a href="#">kPORT_MuxAsGpio</a> : Set as GPIO.</li><li>• <a href="#">kPORT_MuxAlt2</a> : chip-specific.</li><li>• <a href="#">kPORT_MuxAlt3</a> : chip-specific.</li><li>• <a href="#">kPORT_MuxAlt4</a> : chip-specific.</li><li>• <a href="#">kPORT_MuxAlt5</a> : chip-specific.</li><li>• <a href="#">kPORT_MuxAlt6</a> : chip-specific.</li><li>• <a href="#">kPORT_MuxAlt7</a> : chip-specific. : 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 : <a href="#">kPORT_PinDisabledOrAnalog</a>). This function is recommended to use to reset the pin mux</li></ul>

#### 26.6.4 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>• #kPORT_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).</li> <li>• #kPORT_DMAPFallingEdge: DMA request on falling edge(if the DMA requests exit).</li> <li>• #kPORT_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).</li> <li>• #kPORT_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).</li> <li>• #kPORT_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).</li> <li>• #kPORT_FlagEitherEdge : 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>• #kPORT_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).</li> <li>• #kPORT_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit).</li> </ul>

#### 26.6.5 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

## Function Documentation

<i>base</i>	PORT peripheral base pointer.
-------------	-------------------------------

Returns

Current port interrupt status flags, for example, 0x00010001 means the pin 0 and 17 have the interrupt.

### 26.6.6 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 27

## QSPI: Quad Serial Peripheral Interface Driver

### 27.1 Overview

The KSDK provides a peripheral driver for the Quad Serial Peripheral Interface (QSPI) module of Kinetis devices.

QSPI driver includes functional APIs and EDMA transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for QSPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the QSPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. QSPI 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 `qspi_handle_t` as the first parameter. Initialize the handle by calling the [QSPI\\_TransferTxCreateHandleEDMA\(\)](#) or [QSPI\\_TransferRxCreateHandleEDMA\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [QSPI\\_TransferSendEDMA\(\)](#) and [QSPI\\_TransferReceiveEDMA\(\)](#) set up EDMA for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_QSPI_Idle` status.

### Modules

- [QSPI eDMA Driver](#)

### Data Structures

- struct `qspi_dqs_config_t`  
*DQS configure features.* [More...](#)
- struct `qspi_flash_timing_t`  
*Flash timing configuration.* [More...](#)
- struct `qspi_config_t`  
*QSPI configuration structure.* [More...](#)
- struct `qspi_flash_config_t`  
*External flash configuration items.* [More...](#)
- struct `qspi_transfer_t`  
*Transfer structure for QSPI.* [More...](#)

## Overview

### Enumerations

- enum `_status_t` {  
    `kStatus_QSPI_Idle` = MAKE\_STATUS(kStatusGroup\_QSPI, 0),  
    `kStatus_QSPI_Busy` = MAKE\_STATUS(kStatusGroup\_QSPI, 1),  
    `kStatus_QSPI_Error` = MAKE\_STATUS(kStatusGroup\_QSPI, 2) }  
  
    *Status structure of QSPI.*
- enum `qspi_read_area_t` {  
    `kQSPI_ReadAHB` = 0x0U,  
    `kQSPI_ReadIP` }  
  
    *QSPI read data area, from IP FIFO or AHB buffer.*
- enum `qspi_command_seq_t` {  
    `kQSPI_IPSeq` = QuadSPI\_SPTRCLR\_IPPTRC\_MASK,  
    `kQSPI_BufferSeq` = QuadSPI\_SPTRCLR\_BFPTRC\_MASK }  
  
    *QSPI command sequence type.*
- enum `qspi_fifo_t` {  
    `kQSPI_TxFifo` = QuadSPI\_MCR\_CLR\_TXF\_MASK,  
    `kQSPI_RxFifo` = QuadSPI\_MCR\_CLR\_RXF\_MASK,  
    `kQSPI_AllFifo` = QuadSPI\_MCR\_CLR\_TXF\_MASK | QuadSPI\_MCR\_CLR\_RXF\_MASK }  
  
    *QSPI buffer type.*
- enum `qspi_endianness_t` {  
    `kQSPI_64BigEndian` = 0x0U,  
    `kQSPI_32LittleEndian`,  
    `kQSPI_32BigEndian`,  
    `kQSPI_64LittleEndian` }  
  
    *QSPI transfer endianess.*
- enum `_qspi_error_flags` {  
    `kQSPI_DataLearningFail` = QuadSPI\_FR\_DLPFF\_MASK,  
    `kQSPI_TxBufferFill` = QuadSPI\_FR\_TBFF\_MASK,  
    `kQSPI_TxBufferUnderrun` = QuadSPI\_FR\_TBUF\_MASK,  
    `kQSPI_IllegalInstruction` = QuadSPI\_FR\_ILLINE\_MASK,  
    `kQSPI_RxBufferOverflow` = QuadSPI\_FR\_RBOF\_MASK,  
    `kQSPI_RxBufferDrain` = QuadSPI\_FR\_RBDF\_MASK,  
    `kQSPI_AHBSequenceError` = QuadSPI\_FR\_ABSEF\_MASK,  
    `kQSPI_AHBIlegalTransaction` = QuadSPI\_FR\_AITEF\_MASK,  
    `kQSPI_AHBIlegalBurstSize` = QuadSPI\_FR\_AIBSEF\_MASK,  
    `kQSPI_AHBBufferOverflow` = QuadSPI\_FR\_ABOF\_MASK,  
    `kQSPI_IPCommandUsageError` = QuadSPI\_FR\_IUEF\_MASK,  
    `kQSPI_IPCommandTriggerDuringAHBAccess` = QuadSPI\_FR\_IPAEF\_MASK,  
    `kQSPI_IPCommandTriggerDuringIPAccess` = QuadSPI\_FR\_IPIEF\_MASK,  
    `kQSPI_IPCommandTriggerDuringAHBGrant` = QuadSPI\_FR\_IPGEF\_MASK,  
    `kQSPI_IPCommandTransactionFinished` = QuadSPI\_FR\_TFF\_MASK,  
    `kQSPI_FlagAll` = 0x8C83F8D1U }  
  
    *QSPI error flags.*
- enum `_qspi_flags` {

```

kQSPI_DataLearningSamplePoint = QuadSPI_SR_DLPSMP_MASK,
kQSPI_TxBufferFull = QuadSPI_SR_TXFULL_MASK,
kQSPI_TxDMA = QuadSPI_SR_TXDMA_MASK,
kQSPI_TxWatermark = QuadSPI_SR_TXWA_MASK,
kQSPI_TxBufferEnoughData = QuadSPI_SR_TXEDA_MASK,
kQSPI_RxDMA = QuadSPI_SR_RXDMA_MASK,
kQSPI_RxBufferFull = QuadSPI_SR_RXFULL_MASK,
kQSPI_RxWatermark = QuadSPI_SR_RXWE_MASK,
kQSPI_AHB3BufferFull = QuadSPI_SR_AHB3FUL_MASK,
kQSPI_AHB2BufferFull = QuadSPI_SR_AHB2FUL_MASK,
kQSPI_AHB1BufferFull = QuadSPI_SR_AHB1FUL_MASK,
kQSPI_AHB0BufferFull = QuadSPI_SR_AHB0FUL_MASK,
kQSPI_AHB3BufferNotEmpty = QuadSPI_SR_AHB3NE_MASK,
kQSPI_AHB2BufferNotEmpty = QuadSPI_SR_AHB2NE_MASK,
kQSPI_AHB1BufferNotEmpty = QuadSPI_SR_AHB1NE_MASK,
kQSPI_AHB0BufferNotEmpty = QuadSPI_SR_AHB0NE_MASK,
kQSPI_AHBTransactionPending = QuadSPI_SR_AHBTRN_MASK,
kQSPI_AHBCCommandPriorityGranted = QuadSPI_SR_AHBGNT_MASK,
kQSPI_AHBAccess = QuadSPI_SR_AHB_ACC_MASK,
kQSPI_IPAccess = QuadSPI_SR_IP_ACC_MASK,
kQSPI_Busy = QuadSPI_SR_BUSY_MASK,
kQSPI_StateAll = 0xEF897FE7U }

```

*QSPI state bit.*

- enum `_qspi_interrupt_enable` {

```

kQSPI_DataLearningFailInterruptEnable,
kQSPI_TxBufferFillInterruptEnable = QuadSPI_RSER_TBFIE_MASK,
kQSPI_TxBufferUnderrunInterruptEnable = QuadSPI_RSER_TBUIE_MASK,
kQSPI_IllegalInstructionInterruptEnable,
kQSPI_RxBufferOverflowInterruptEnable = QuadSPI_RSER_RBOIE_MASK,
kQSPI_RxBufferDrainInterruptEnable = QuadSPI_RSER_RBDIE_MASK,
kQSPI_AHBSequenceErrorInterruptEnable = QuadSPI_RSER_ABSEIE_MASK,
kQSPI_AHBIlegalTransactionInterruptEnable,
kQSPI_AHBIlegalBurstSizeInterruptEnable,
kQSPI_AHBBufferOverflowInterruptEnable = QuadSPI_RSER_ABOIE_MASK,
kQSPI_IPCommandUsageErrorInterruptEnable = QuadSPI_RSER_IUEIE_MASK,
kQSPI_IPCommandTriggerDuringAHBAccessInterruptEnable,
kQSPI_IPCommandTriggerDuringIPAccessInterruptEnable,
kQSPI_IPCommandTriggerDuringAHBGrantInterruptEnable,
kQSPI_IPCommandTransactionFinishedInterruptEnable,
kQSPI_AllInterruptEnable = 0x8C83F8D1U }

```

*QSPI interrupt enable.*

- enum `_qspi_dma_enable` {

```

kQSPI_TxBufferFillDMAEnable = QuadSPI_RSER_TBFDE_MASK,
kQSPI_RxBufferDrainDMAEnable = QuadSPI_RSER_RBDDE_MASK,
kQSPI_AllDDMAEnable = QuadSPI_RSER_TBFDE_MASK | QuadSPI_RSER_RBDDE_MASK

```

## Overview

- }
- QSPI DMA request flag.*
- enum `qspi_dqs_phrase_shift_t` {  
    kQSPI\_DQSNoPhraseShift = 0x0U,  
    kQSPI\_DQSPhraseShift45Degree,  
    kQSPI\_DQSPhraseShift90Degree,  
    kQSPI\_DQSPhraseShift135Degree }
- Phrase shift number for DQS mode.*

## Driver version

- #define `FSL_QSPI_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 1)`)  
*I2C driver version 2.0.1.*

## Initialization and deinitialization

- void `QSPI_Init` (QuadSPI\_Type \*base, `qspi_config_t` \*config, uint32\_t srcClock\_Hz)  
*Initializes the QSPI module and internal state.*
- void `QSPI_GetDefaultQspiConfig` (`qspi_config_t` \*config)  
*Gets default settings for QSPI.*
- void `QSPI_Deinit` (QuadSPI\_Type \*base)  
*Deinitializes the QSPI module.*
- void `QSPI_SetFlashConfig` (QuadSPI\_Type \*base, `qspi_flash_config_t` \*config)  
*Configures the serial flash parameter.*
- void `QSPI_SoftwareReset` (QuadSPI\_Type \*base)  
*Software reset for the QSPI logic.*
- static void `QSPI_Enable` (QuadSPI\_Type \*base, bool enable)  
*Enables or disables the QSPI module.*

## Status

- static uint32\_t `QSPI_GetStatusFlags` (QuadSPI\_Type \*base)  
*Gets the state value of QSPI.*
- static uint32\_t `QSPI_GetErrorStatusFlags` (QuadSPI\_Type \*base)  
*Gets QSPI error status flags.*
- static void `QSPI_ClearErrorFlag` (QuadSPI\_Type \*base, uint32\_t mask)  
*Clears the QSPI error flags.*

## Interrupts

- static void `QSPI_EnableInterrupts` (QuadSPI\_Type \*base, uint32\_t mask)  
*Enables the QSPI interrupts.*
- static void `QSPI_DisableInterrupts` (QuadSPI\_Type \*base, uint32\_t mask)  
*Disables the QSPI interrupts.*

## DMA Control

- static void `QSPI_EnableDMA` (QuadSPI\_Type \*base, uint32\_t mask, bool enable)  
*Enables the QSPI DMA source.*
- static uint32\_t `QSPI_GetTxDataRegisterAddress` (QuadSPI\_Type \*base)

- `uint32_t QSPI_GetRxDataRegisterAddress (QuadSPI_Type *base)`  
*Gets the Rx data register address used for DMA operation.*

## Bus Operations

- `static void QSPI_SetIPCommandAddress (QuadSPI_Type *base, uint32_t addr)`  
*Sets the IP command address.*
- `static void QSPI_SetIPCommandSize (QuadSPI_Type *base, uint32_t size)`  
*Sets the IP command size.*
- `void QSPI_ExecuteIPCommand (QuadSPI_Type *base, uint32_t index)`  
*Executes IP commands located in LUT table.*
- `void QSPI_ExecuteAHBCommand (QuadSPI_Type *base, uint32_t index)`  
*Executes AHB commands located in LUT table.*
- `static void QSPI_EnableIPParallelMode (QuadSPI_Type *base, bool enable)`  
*Enables/disables the QSPI IP command parallel mode.*
- `static void QSPI_EnableAHBParallelMode (QuadSPI_Type *base, bool enable)`  
*Enables/disables the QSPI AHB command parallel mode.*
- `void QSPI_UpdateLUT (QuadSPI_Type *base, uint32_t index, uint32_t *cmd)`  
*Updates the LUT table.*
- `static void QSPI_ClearFifo (QuadSPI_Type *base, uint32_t mask)`  
*Clears the QSPI FIFO logic.*
- `static void QSPI_ClearCommandSequence (QuadSPI_Type *base, qspi_command_seq_t seq)`  
*@ brief Clears the command sequence for the IP/buffer command.*
- `void QSPI_SetReadDataArea (QuadSPI_Type *base, qspi_read_area_t area)`  
*@ brief Set the RX buffer readout area.*
- `void QSPI_WriteBlocking (QuadSPI_Type *base, uint32_t *buffer, size_t size)`  
*Sends a buffer of data bytes using a blocking method.*
- `static void QSPI_WriteData (QuadSPI_Type *base, uint32_t data)`  
*Writes data into FIFO.*
- `void QSPI_ReadBlocking (QuadSPI_Type *base, uint32_t *buffer, size_t size)`  
*Receives a buffer of data bytes using a blocking method.*
- `uint32_t QSPI_ReadData (QuadSPI_Type *base)`  
*Receives data from data FIFO.*

## Transactional

- `static void QSPI_TransferSendBlocking (QuadSPI_Type *base, qspi_transfer_t *xfer)`  
*Writes data to the QSPI transmit buffer.*
- `static void QSPI_TransferReceiveBlocking (QuadSPI_Type *base, qspi_transfer_t *xfer)`  
*Reads data from the QSPI receive buffer in polling way.*

## 27.2 Data Structure Documentation

### 27.2.1 struct qspi\_dqs\_config\_t

#### Data Fields

- `uint32_t portADelayTapNum`  
*Delay chain tap number selection for QSPI port A DQS.*

## Data Structure Documentation

- `uint32_t portBDelayTapNum`  
*Delay chain tap number selection for QSPI port B DQS.*
- `qspi_dqs_phrase_shift_t shift`  
*Phase shift for internal DQS generation.*
- `bool enableDQSClkInverse`  
*Enable inverse clock for internal DQS generation.*
- `bool enableDQSPadLoopback`  
*Enable DQS loop back from DQS pad.*
- `bool enableDQSLoopback`  
*Enable DQS loop back.*

### 27.2.2 struct qspi\_flash\_timing\_t

#### Data Fields

- `uint32_t dataHoldTime`  
*Serial flash data in hold time.*
- `uint32_t CSHoldTime`  
*Serial flash CS hold time in terms of serial flash clock cycles.*
- `uint32_t CSSetupTime`  
*Serial flash CS setup time in terms of serial flash clock cycles.*

### 27.2.3 struct qspi\_config\_t

#### Data Fields

- `uint32_t clockSource`  
*Clock source for QSPI module.*
- `uint32_t baudRate`  
*Serial flash clock baud rate.*
- `uint8_t txWatermark`  
*QSPI transmit watermark value.*
- `uint8_t rxWatermark`  
*QSPI receive watermark value.*
- `uint32_t AHBbufferSize` [FSL\_FEATURE\_QSPI\_AHB\_BUFFER\_COUNT]  
*AHB buffer size.*
- `uint8_t AHBbufferMaster` [FSL\_FEATURE\_QSPI\_AHB\_BUFFER\_COUNT]  
*AHB buffer master.*
- `bool enableAHBbuffer3AllMaster`  
*Is AHB buffer3 for all master.*
- `qspi_read_area_t area`  
*Which area Rx data readout.*
- `bool enableQspi`  
*Enable QSPI after initialization.*

### 27.2.3.0.5.7 Field Documentation

27.2.3.0.5.7.1 `uint8_t qspi_config_t::rxWatermark`

27.2.3.0.5.7.2 `uint32_t qspi_config_t::AHBbufferSize[FSL_FEATURE_QSPI_AHB_BUFFER_COUNT]`

27.2.3.0.5.7.3 `uint8_t qspi_config_t::AHBbufferMaster[FSL_FEATURE_QSPI_AHB_BUFFER_COUNT]`

27.2.3.0.5.7.4 `bool qspi_config_t::enableAHBbuffer3AllMaster`

## 27.2.4 struct qspi\_flash\_config\_t

### Data Fields

- `uint32_t flashA1Size`  
*Flash A1 size.*
- `uint32_t flashA2Size`  
*Flash A2 size.*
- `uint32_t flashB1Size`  
*Flash B1 size.*
- `uint32_t flashB2Size`  
*Flash B2 size.*
- `uint32_t lookuptable [FSL_FEATURE_QSPI_LUT_DEPTH]`  
*Flash command in LUT.*
- `uint32_t dataHoldTime`  
*Data line hold time.*
- `uint32_t CSHoldTime`  
*CS line hold time.*
- `uint32_t CSSetupTime`  
*CS line setup time.*
- `uint32_t columnSpace`  
*Column space size.*
- `uint32_t dataLearnValue`  
*Data Learn value if enable data learn.*
- `qspi_endianess_t endian`  
*Flash data endianess.*
- `bool enableWordAddress`  
*If enable word address.*

## Enumeration Type Documentation

### 27.2.4.0.5.8 Field Documentation

27.2.4.0.5.8.1 `uint32_t qspi_flash_config_t::dataHoldTime`

27.2.4.0.5.8.2 `qspi_endianness_t qspi_flash_config_t::endian`

27.2.4.0.5.8.3 `bool qspi_flash_config_t::enableWordAddress`

### 27.2.5 `struct qspi_transfer_t`

#### Data Fields

- `uint32_t * data`  
*Pointer to data to transmit.*
- `size_t dataSize`  
*Bytes to be transmit.*

### 27.3 Macro Definition Documentation

27.3.1 `#define FSL_QSPI_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))`

### 27.4 Enumeration Type Documentation

#### 27.4.1 `enum _status_t`

Enumerator

*kStatus\_QSPI\_Idle* QSPI is in idle state.

*kStatus\_QSPI\_Busy* QSPI is busy.

*kStatus\_QSPI\_Error* Error occurred during QSPI transfer.

#### 27.4.2 `enum qspi_read_area_t`

Enumerator

*kQSPI\_ReadAHB* QSPI read from AHB buffer.

*kQSPI\_ReadIP* QSPI read from IP FIFO.

#### 27.4.3 `enum qspi_command_seq_t`

Enumerator

*kQSPI\_IPSeq* IP command sequence.

*kQSPI\_BufferSeq* Buffer command sequence.

#### 27.4.4 enum qspi\_fifo\_t

Enumerator

- kQSPI\_TxFifo* QSPI Tx FIFO.
- kQSPI\_RxFifo* QSPI Rx FIFO.
- kQSPI\_AllFifo* QSPI all FIFO, including Tx and Rx.

#### 27.4.5 enum qspi\_endianness\_t

Enumerator

- kQSPI\_64BigEndian* 64 bits big endian
- kQSPI\_32LittleEndian* 32 bit little endian
- kQSPI\_32BigEndian* 32 bit big endian
- kQSPI\_64LittleEndian* 64 bit little endian

#### 27.4.6 enum \_qspi\_error\_flags

Enumerator

- kQSPI\_DataLearningFail* Data learning pattern failure flag.
- kQSPI\_TxBufferFill* Tx buffer fill flag.
- kQSPI\_TxBufferUnderrun* Tx buffer underrun flag.
- kQSPI\_IllegalInstruction* Illegal instruction error flag.
- kQSPI\_RxBufferOverflow* Rx buffer overflow flag.
- kQSPI\_RxBufferDrain* Rx buffer drain flag.
- kQSPI\_AHBSquenceError* AHB sequence error flag.
- kQSPI\_AHBIllegalTransaction* AHB illegal transaction error flag.
- kQSPI\_AHBIllegalBurstSize* AHB illegal burst error flag.
- kQSPI\_AHBBufferOverflow* AHB buffer overflow flag.
- kQSPI\_IPCommandUsageError* IP command usage error flag.
- kQSPI\_IPCommandTriggerDuringAHBAccess* IP command trigger during AHB access error.
- kQSPI\_IPCommandTriggerDuringIPAccess* IP command trigger cannot be executed.
- kQSPI\_IPCommandTriggerDuringAHBGrant* IP command trigger during AHB grant error.
- kQSPI\_IPCommandTransactionFinished* IP command transaction finished flag.
- kQSPI\_FlagAll* All error flag.

#### 27.4.7 enum \_qspi\_flags

Enumerator

- kQSPI\_DataLearningSamplePoint* Data learning sample point.

## Enumeration Type Documentation

*kQSPI\_TxBufferFull* Tx buffer full flag.  
*kQSPI\_TxDMA* Tx DMA is requested or running.  
*kQSPI\_TxWatermark* Tx buffer watermark available.  
*kQSPI\_TxBufferEnoughData* Tx buffer enough data available.  
*kQSPI\_RxDMA* Rx DMA is requesting or running.  
*kQSPI\_RxBufferFull* Rx buffer full.  
*kQSPI\_RxWatermark* Rx buffer watermark exceeded.  
*kQSPI\_AHB3BufferFull* AHB buffer 3 full.  
*kQSPI\_AHB2BufferFull* AHB buffer 2 full.  
*kQSPI\_AHB1BufferFull* AHB buffer 1 full.  
*kQSPI\_AHB0BufferFull* AHB buffer 0 full.  
*kQSPI\_AHB3BufferNotEmpty* AHB buffer 3 not empty.  
*kQSPI\_AHB2BufferNotEmpty* AHB buffer 2 not empty.  
*kQSPI\_AHB1BufferNotEmpty* AHB buffer 1 not empty.  
*kQSPI\_AHB0BufferNotEmpty* AHB buffer 0 not empty.  
*kQSPI\_AHBTransactionPending* AHB access transaction pending.  
*kQSPI\_AHBCCommandPriorityGranted* AHB command priority granted.  
*kQSPI\_AHBAccess* AHB access.  
*kQSPI\_IPAccess* IP access.  
*kQSPI\_Busy* Module busy.  
*kQSPI\_StateAll* All flags.

### 27.4.8 enum \_qspi\_interrupt\_enable

Enumerator

*kQSPI\_DataLearningFailInterruptEnable* Data learning pattern failure interrupt enable.  
*kQSPI\_TxBufferFillInterruptEnable* Tx buffer fill interrupt enable.  
*kQSPI\_TxBufferUnderrunInterruptEnable* Tx buffer underrun interrupt enable.  
*kQSPI\_IllegalInstructionInterruptEnable* Illegal instruction error interrupt enable.  
*kQSPI\_RxBufferOverflowInterruptEnable* Rx buffer overflow interrupt enable.  
*kQSPI\_RxBufferDrainInterruptEnable* Rx buffer drain interrupt enable.  
*kQSPI\_AHBSequenceErrorInterruptEnable* AHB sequence error interrupt enable.  
*kQSPI\_AHBIlegalTransactionInterruptEnable* AHB illegal transaction error interrupt enable.  
*kQSPI\_AHBIlegalBurstSizeInterruptEnable* AHB illegal burst error interrupt enable.  
*kQSPI\_AHBBufferOverflowInterruptEnable* AHB buffer overflow interrupt enable.  
*kQSPI\_IPCommandUsageErrorInterruptEnable* IP command usage error interrupt enable.  
*kQSPI\_IPCommandTriggerDuringAHBAccesInterruptEnable* IP command trigger during AHB access error.  
*kQSPI\_IPCommandTriggerDuringIPAccessInterruptEnable* IP command trigger cannot be executed.  
*kQSPI\_IPCommandTriggerDuringAHBGrantInterruptEnable* IP command trigger during AHB grant error.

***kQSPI\_IPCommandTransactionFinishedInterruptEnable*** IP command transaction finished interrupt enable.

***kQSPI\_AllInterruptEnable*** All error interrupt enable.

#### 27.4.9 enum \_qspi\_dma\_enable

Enumerator

***kQSPI\_TxBufferFillDMAEnable*** Tx buffer fill DMA.

***kQSPI\_RxBufferDrainDMAEnable*** Rx buffer drain DMA.

***kQSPI\_AllDDMAEnable*** All DMA source.

#### 27.4.10 enum qspi\_dqs\_phrase\_shift\_t

Enumerator

***kQSPI\_DQSNoPhraseShift*** No phase shift.

***kQSPI\_DQSPhraseShift45Degree*** Select 45 degree phase shift.

***kQSPI\_DQSPhraseShift90Degree*** Select 90 degree phase shift.

***kQSPI\_DQSPhraseShift135Degree*** Select 135 degree phase shift.

### 27.5 Function Documentation

#### 27.5.1 void QSPI\_Init ( QuadSPI\_Type \* *base*, qspi\_config\_t \* *config*, uint32\_t *srcClock\_Hz* )

This function enables the clock for QSPI and also configures the QSPI with the input configure parameters. Users should call this function before any QSPI operations.

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>config</i>	QSPI configure structure.
<i>srcClock_Hz</i>	QSPI source clock frequency in Hz.

#### 27.5.2 void QSPI\_GetDefaultQspiConfig ( qspi\_config\_t \* *config* )

## Function Documentation

Parameters

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

### 27.5.3 void QSPI\_Deinit ( QuadSPI\_Type \* *base* )

Clears the QSPI state and QSPI module registers.

Parameters

<i>base</i>	Pointer to QuadSPI Type.
-------------	--------------------------

### 27.5.4 void QSPI\_SetFlashConfig ( QuadSPI\_Type \* *base*, qspi\_flash\_config\_t \* *config* )

This function configures the serial flash relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the QSPI features.

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>config</i>	Flash configuration parameters.

### 27.5.5 void QSPI\_SoftwareReset ( QuadSPI\_Type \* *base* )

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

Parameters

<i>base</i>	Pointer to QuadSPI Type.
-------------	--------------------------

### 27.5.6 static void QSPI\_Enable ( QuadSPI\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>enable</i>	True means enable QSPI, false means disable.

### 27.5.7 static uint32\_t QSPI\_GetStatusFlags ( QuadSPI\_Type \* *base* ) [inline], [static]

Parameters

<i>base</i>	Pointer to QuadSPI Type.
-------------	--------------------------

Returns

status flag, use status flag to AND [\\_qspi\\_flags](#) could get the related status.

### 27.5.8 static uint32\_t QSPI\_GetErrorStatusFlags ( QuadSPI\_Type \* *base* ) [inline], [static]

Parameters

<i>base</i>	Pointer to QuadSPI Type.
-------------	--------------------------

Returns

status flag, use status flag to AND [\\_qspi\\_error\\_flags](#) could get the related status.

### 27.5.9 static void QSPI\_ClearErrorFlag ( QuadSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	Pointer to QuadSPI Type.
-------------	--------------------------

## Function Documentation

<i>mask</i>	Which kind of QSPI flags to be cleared, a combination of _qspi_error_flags.
-------------	---

**27.5.10 static void QSPI\_EnableInterrupts ( QuadSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>mask</i>	QSPI interrupt source.

**27.5.11 static void QSPI\_DisableInterrupts ( QuadSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>mask</i>	QSPI interrupt source.

**27.5.12 static void QSPI\_EnableDMA ( QuadSPI\_Type \* *base*, uint32\_t *mask*, bool *enable* ) [inline], [static]**

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>mask</i>	QSPI DMA source.
<i>enable</i>	True means enable DMA, false means disable.

**27.5.13 static uint32\_t QSPI\_GetTxDataRegisterAddress ( QuadSPI\_Type \* *base* ) [inline], [static]**

It is used for DMA operation.

Parameters

<i>base</i>	Pointer to QuadSPI Type.
-------------	--------------------------

Returns

QSPI Tx data register address.

#### 27.5.14 **uint32\_t QSPI\_GetRxDataRegisterAddress ( QuadSPI\_Type \* *base* )**

This function returns the Rx data register address or Rx buffer address according to the Rx read area settings.

Parameters

<i>base</i>	Pointer to QuadSPI Type.
-------------	--------------------------

Returns

QSPI Rx data register address.

#### 27.5.15 **static void QSPI\_SetIPCommandAddress ( QuadSPI\_Type \* *base*, uint32\_t *addr* ) [inline], [static]**

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>addr</i>	IP command address.

#### 27.5.16 **static void QSPI\_SetIPCommandSize ( QuadSPI\_Type \* *base*, uint32\_t *size* ) [inline], [static]**

Parameters

## Function Documentation

<i>base</i>	Pointer to QuadSPI Type.
<i>size</i>	IP command size.

**27.5.17 void QSPI\_ExecuteIPCommand ( QuadSPI\_Type \* *base*, uint32\_t *index* )**

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>index</i>	IP command located in which LUT table index.

**27.5.18 void QSPI\_ExecuteAHBCommand ( QuadSPI\_Type \* *base*, uint32\_t *index* )**

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>index</i>	AHB command located in which LUT table index.

**27.5.19 static void QSPI\_EnableIPParallelMode ( QuadSPI\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>enable</i>	True means enable parallel mode, false means disable parallel mode.

**27.5.20 static void QSPI\_EnableAHBParallelMode ( QuadSPI\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>enable</i>	True means enable parallel mode, false means disable parallel mode.

**27.5.21 void QSPI\_UpdateLUT ( QuadSPI\_Type \* *base*, uint32\_t *index*, uint32\_t \* *cmd* )**

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>index</i>	Which LUT index needs to be located. It should be an integer divided by 4.
<i>cmd</i>	Command sequence array.

**27.5.22 static void QSPI\_ClearFifo ( QuadSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>mask</i>	Which kind of QSPI FIFO to be cleared.

**27.5.23 static void QSPI\_ClearCommandSequence ( QuadSPI\_Type \* *base*, qspi\_command\_seq\_t *seq* ) [inline], [static]**

This function can reset the command sequence.

Parameters

<i>base</i>	QSPI base address.
<i>seq</i>	Which command sequence need to reset, IP command, buffer command or both.

**27.5.24 void QSPI\_SetReadDataArea ( QuadSPI\_Type \* *base*, qspi\_read\_area\_t *area* )**

This function can set the RX buffer readout, from AHB bus or IP Bus.

## Function Documentation

Parameters

<i>base</i>	QSPI base address.
<i>area</i>	QSPI Rx buffer readout area. AHB bus buffer or IP bus buffer.

**27.5.25 void QSPI\_WriteBlocking ( QuadSPI\_Type \* *base*, uint32\_t \* *buffer*, size\_t *size* )**

Note

This function blocks via polling until all bytes have been sent.

Parameters

<i>base</i>	QSPI base pointer
<i>buffer</i>	The data bytes to send
<i>size</i>	The number of data bytes to send

**27.5.26 static void QSPI\_WriteData ( QuadSPI\_Type \* *base*, uint32\_t *data* )  
[inline], [static]**

Parameters

<i>base</i>	QSPI base pointer
<i>data</i>	The data bytes to send

**27.5.27 void QSPI\_ReadBlocking ( QuadSPI\_Type \* *base*, uint32\_t \* *buffer*, size\_t *size* )**

Note

This function blocks via polling until all bytes have been sent.

Parameters

<i>base</i>	QSPI base pointer
<i>buffer</i>	The data bytes to send
<i>size</i>	The number of data bytes to receive

### 27.5.28 `uint32_t QSPI_ReadData ( QuadSPI_Type * base )`

Parameters

<i>base</i>	QSPI base pointer
-------------	-------------------

Returns

The data in the FIFO.

### 27.5.29 `static void QSPI_TransferSendBlocking ( QuadSPI_Type * base, qspi_transfer_t * xfer ) [inline], [static]`

This function writes a continuous data to the QSPI transmit FIFO. This function is a block function and can return only when finished. This function uses polling methods.

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>xfer</i>	QSPI transfer structure.

### 27.5.30 `static void QSPI_TransferReceiveBlocking ( QuadSPI_Type * base, qspi_transfer_t * xfer ) [inline], [static]`

This function reads continuous data from the QSPI receive buffer/FIFO. This function is a blocking function and can return only when finished. This function uses polling methods.

Parameters

## Function Documentation

<i>base</i>	Pointer to QuadSPI Type.
<i>xfer</i>	QSPI transfer structure.

## 27.6 QSPI eDMA Driver

### 27.6.1 Overview

#### Data Structures

- struct `qspi_edma_handle_t`

*QSPI DMA transfer handle, users should not touch the content of the handle. [More...](#)*

#### TypeDefs

- typedef void(\* `qspi_edma_callback_t`)(QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle, status\_t status, void \*userData)

*QSPI eDMA transfer callback function for finish and error.*

#### eDMA Transactional

- void `QSPI_TransferTxCreateHandleEDMA` (QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle, `qspi_edma_callback_t` callback, void \*userData, `edma_handle_t` \*dmaHandle)  
*Initializes the QSPI handle for send which is used in transactional functions and set the callback.*
- void `QSPI_TransferRxCreateHandleEDMA` (QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle, `qspi_edma_callback_t` callback, void \*userData, `edma_handle_t` \*dmaHandle)  
*Initializes the QSPI handle for receive which is used in transactional functions and set the callback.*
- status\_t `QSPI_TransferSendEDMA` (QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle, `qspi_transfer_t` \*xfer)  
*Transfers QSPI data using an eDMA non-blocking method.*
- status\_t `QSPI_TransferReceiveEDMA` (QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle, `qspi_transfer_t` \*xfer)  
*Receives data using an eDMA non-blocking method.*
- void `QSPI_TransferAbortSendEDMA` (QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle)  
*Aborts the sent data using eDMA.*
- void `QSPI_TransferAbortReceiveEDMA` (QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle)  
*Aborts the receive data using eDMA.*
- status\_t `QSPI_TransferGetSendCountEDMA` (QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets the transferred counts of send.*
- status\_t `QSPI_TransferGetReceiveCountEDMA` (QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets the status of the receive transfer.*

### 27.6.2 Data Structure Documentation

#### 27.6.2.1 struct \_qspi\_edma\_handle

##### Data Fields

- `edma_handle_t * dmaHandle`  
*eDMA handler for QSPI send*
- `size_t transferSize`  
*Bytes need to transfer.*
- `uint8_t count`  
*The transfer data count in a DMA request.*
- `uint32_t state`  
*Internal state for QSPI eDMA transfer.*
- `qspi_edma_callback_t callback`  
*Callback for users while transfer finish or error occurred.*
- `void * userData`  
*User callback parameter.*

#### 27.6.2.1.0.9 Field Documentation

##### 27.6.2.1.0.9.1 size\_t qspi\_edma\_handle\_t::transferSize

### 27.6.3 Function Documentation

#### 27.6.3.1 void QSPI\_TransferTxCreateHandleEDMA ( `QuadSPI_Type * base,` `qspi_edma_handle_t * handle, qspi_edma_callback_t callback, void * userData,` `edma_handle_t * dmaHandle )`

Parameters

<code>base</code>	QSPI peripheral base address
<code>handle</code>	Pointer to <code>qspi_edma_handle_t</code> structure
<code>callback</code>	QSPI callback, NULL means no callback.
<code>userData</code>	User callback function data.
<code>rxDmaHandle</code>	User requested eDMA handle for eDMA transfer

#### 27.6.3.2 void QSPI\_TransferRxCreateHandleEDMA ( `QuadSPI_Type * base,` `qspi_edma_handle_t * handle, qspi_edma_callback_t callback, void * userData,` `edma_handle_t * dmaHandle )`

Parameters

<i>base</i>	QSPI peripheral base address
<i>handle</i>	Pointer to qspi_edma_handle_t structure
<i>callback</i>	QSPI callback, NULL means no callback.
<i>userData</i>	User callback function data.
<i>rxDmaHandle</i>	User requested eDMA handle for eDMA transfer

### 27.6.3.3 status\_t QSPI\_TransferSendEDMA ( QuadSPI\_Type \* *base*, qspi\_edma\_handle\_t \* *handle*, qspi\_transfer\_t \* *xfer* )

This function writes data to the QSPI transmit FIFO. This function is non-blocking.

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>handle</i>	Pointer to qspi_edma_handle_t structure
<i>xfer</i>	QSPI transfer structure.

### 27.6.3.4 status\_t QSPI\_TransferReceiveEDMA ( QuadSPI\_Type \* *base*, qspi\_edma\_handle\_t \* *handle*, qspi\_transfer\_t \* *xfer* )

This function receive data from the QSPI receive buffer/FIFO. This function is non-blocking.

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>handle</i>	Pointer to qspi_edma_handle_t structure
<i>xfer</i>	QSPI transfer structure.

### 27.6.3.5 void QSPI\_TransferAbortSendEDMA ( QuadSPI\_Type \* *base*, qspi\_edma\_handle\_t \* *handle* )

This function aborts the sent data using eDMA.

## QSPI eDMA Driver

Parameters

<i>base</i>	QSPI peripheral base address.
<i>handle</i>	Pointer to qspi_edma_handle_t structure

**27.6.3.6 void QSPI\_TransferAbortReceiveEDMA ( QuadSPI\_Type \* *base*,  
qspi\_edma\_handle\_t \* *handle* )**

This function abort receive data which using eDMA.

Parameters

<i>base</i>	QSPI peripheral base address.
<i>handle</i>	Pointer to qspi_edma_handle_t structure

**27.6.3.7 status\_t QSPI\_TransferGetSendCountEDMA ( QuadSPI\_Type \* *base*,  
qspi\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>handle</i>	Pointer to qspi_edma_handle_t structure.
<i>count</i>	Bytes sent.

Return values

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

**27.6.3.8 status\_t QSPI\_TransferGetReceiveCountEDMA ( QuadSPI\_Type \* *base*,  
qspi\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>handle</i>	Pointer to qspi_edma_handle_t structure
<i>count</i>	Bytes received.

Return values

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



# Chapter 28

## RCM: Reset Control Module Driver

### 28.1 Overview

The KSDK provides a Peripheral driver for the Reset Control Module (RCM) module of Kinetis devices.

### Data Structures

- struct `rcm_reset_pin_filter_config_t`  
*Reset pin filter configuration.* [More...](#)

### Enumerations

- enum `rcm_reset_source_t` {  
    `kRCM_SourceLvd` = RCM\_SRS0\_LVD\_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_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, 1))  
*RCM driver version 2.0.1.*

### Reset Control Module APIs

- static uint32\_t `RCM_GetPreviousResetSources` (RCM\_Type \*base)  
*Gets the reset source status which caused a previous reset.*
- void `RCM_ConfigureResetPinFilter` (RCM\_Type \*base, const `rcm_reset_pin_filter_config_t` \*config)  
*Configures the reset pin filter.*

## Enumeration Type Documentation

### 28.2 Data Structure Documentation

#### 28.2.1 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.*

##### 28.2.1.0.0.10 Field Documentation

###### 28.2.1.0.0.10.1 bool rcm\_reset\_pin\_filter\_config\_t::enableFilterInStop

###### 28.2.1.0.0.10.2 rcm\_run\_wait\_filter\_mode\_t rcm\_reset\_pin\_filter\_config\_t::filterInRunWait

###### 28.2.1.0.0.10.3 uint8\_t rcm\_reset\_pin\_filter\_config\_t::busClockFilterCount

### 28.3 Macro Definition Documentation

#### 28.3.1 #define FSL\_RCM\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

### 28.4 Enumeration Type Documentation

#### 28.4.1 enum rcm\_reset\_source\_t

Enumerator

- kRCM\_SourceLvd* Low-voltage detect 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\_SourceSackerr* Parameter could get all reset flags.

#### 28.4.2 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.

## 28.5 Function Documentation

### 28.5.1 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.

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.

### 28.5.2 void RCM\_ConfigureResetPinFilter ( RCM\_Type \* *base*, const rcm\_reset\_pin\_filter\_config\_t \* *config* )

This function sets the reset pin filter including the filter source, filter width, and so on.

Parameters

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

## Function Documentation

# Chapter 29

## RTC: Real Time Clock

### 29.1 Overview

The KSDK provides a driver for the Real Time Clock (RTC) of Kinetis devices.

### 29.2 Function groups

The RTC driver supports operating the module as a time counter.

#### 29.2.1 Initialization and deinitialization

The function [RTC\\_Init\(\)](#) initializes the RTC with specified configurations. The function [RTC\\_GetDefaultConfig\(\)](#) gets the default configurations.

The function [RTC\\_Deinit\(\)](#) disables the RTC timer and disables the module clock.

#### 29.2.2 Set & Get Datetime

The function [RTC\\_SetDatetime\(\)](#) sets the timer period in seconds. User passes in the details in date & time format by using the below data structure.

```
typedef struct _rtc_datetime
{
    uint16_t year;
    uint8_t month;
    uint8_t day;
    uint8_t hour;
    uint8_t minute;
    uint8_t second;
} rtc_datetime_t;
```

The function [RTC\\_GetDatetime\(\)](#) reads the current timer value in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

#### 29.2.3 Set & Get Alarm

The function [RTC\\_SetAlarm\(\)](#) sets the alarm time period in seconds. User passes in the details in date & time format by using the datetime data structure.

The function [RTC\\_GetAlarm\(\)](#) reads the alarm time in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

## Typical use case

### 29.2.4 Start & Stop timer

The function [RTC\\_StartTimer\(\)](#) starts the RTC time counter.

The function [RTC\\_StopTimer\(\)](#) stops the RTC time counter.

### 29.2.5 Status

Provides functions to get and clear the RTC status.

### 29.2.6 Interrupt

Provides functions to enable/disable RTC interrupts and get current enabled interrupts.

### 29.2.7 RTC Oscillator

Some SoC's allow control of the RTC oscillator through the RTC module.

The function [RTC\\_SetOscCapLoad\(\)](#) allows the user to modify the capacitor load configuration of the RTC oscillator.

### 29.2.8 Monotonic Counter

Some SoC's have a 64-bit Monotonic counter available in the RTC module.

The function [RTC\\_SetMonotonicCounter\(\)](#) writes a 64-bit to the counter.

The function [RTC\\_GetMonotonicCounter\(\)](#) reads the monotonic counter and returns the 64-bit counter value to the user.

The function [RTC\\_IncrementMonotonicCounter\(\)](#) increments the Monotonic Counter by one.

## 29.3 Typical use case

### 29.3.1 RTC tick example

Example to set the RTC current time and trigger an alarm.

```
int main(void)
{
    uint32_t sec;
    uint32_t currSeconds;
    rtc_datetime_t date;
    rtc_config_t rtcConfig;

    /* Board pin, clock, debug console init */

    /* Set the RTC current time and trigger an alarm */

    /* ... */

    /* Start the RTC timer */
    RTC_StartTimer();

    /* ... */

    /* Stop the RTC timer */
    RTC_StopTimer();
}
```

```

BOARD_InitHardware();
/* Init RTC */
RTC_GetDefaultConfig(&rtcConfig);
RTC_Init(RTC, &rtcConfig);
/* Select RTC clock source */
BOARD_SetRtcClockSource();

PRINTF("RTC example: set up time to wake up an alarm\r\n");

/* Set a start date time and start RT */
date.year = 2014U;
date.month = 12U;
date.day = 25U;
date.hour = 19U;
date.minute = 0;
date.second = 0;

/* RTC time counter has to be stopped before setting the date & time in the TSR register */
RTC_StopTimer(RTC);

/* Set RTC time to default */
RTC_SetDatetime(RTC, &date);

/* Enable RTC alarm interrupt */
RTC_EnableInterrupts(RTC, kRTC_AlarmInterruptEnable);

/* Enable at the NVIC */
EnableIRQ(RTC IRQn);

/* Start the RTC time counter */
RTC_StartTimer(RTC);

/* This loop will set the RTC alarm */
while (1)
{
    busyWait = true;
    /* Get date time */
    RTC_GetDatetime(RTC, &date);

    /* print default time */
    PRINTF("Current datetime: %04hd-%02hd-%02hd %02hd:%02hd:%02hd\r\n", date.
year, date.month, date.day, date.hour,
        date.minute, date.second);

    /* Get alarm time from user */
    sec = 0;
    PRINTF("Please input the number of second to wait for alarm \r\n");
    PRINTF("The second must be positive value\r\n");
    while (sec < 1)
    {
        SCANF("%d", &sec);
    }

    /* Read the RTC seconds register to get current time in seconds */
    currSeconds = RTC->TSR;

    /* Add alarm seconds to current time */
    currSeconds += sec;

    /* Set alarm time in seconds */
    RTC->TAR = currSeconds;

    /* Get alarm time */
    RTC_GetAlarm(RTC, &date);

    /* Print alarm time */
    PRINTF("Alarm will occur at: %04hd-%02hd-%02hd %02hd:%02hd:%02hd\r\n", date.
year, date.month, date.day,

```

## Typical use case

```
    date.hour, date.minute, date.second);

    /* Wait until alarm occurs */
    while (busyWait)
    {
    }

    PRINTF("\r\n Alarm occurs !!!! ");
}

}
```

## Data Structures

- struct `rtc_datetime_t`  
*Structure is used to hold the date and time.* [More...](#)
- struct `rtc_config_t`  
*RTC config structure.* [More...](#)

## Enumerations

- enum `rtc_interrupt_enable_t` {  
    kRTC\_TimeInvalidInterruptEnable = RTC\_IER\_TIE\_MASK,  
    kRTC\_TimeOverflowInterruptEnable = RTC\_IER\_TOIE\_MASK,  
    kRTC\_AlarmInterruptEnable = RTC\_IER\_TAIE\_MASK,  
    kRTC\_SecondsInterruptEnable = RTC\_IER\_TSIE\_MASK }  
*List of RTC interrupts.*
- enum `rtc_status_flags_t` {  
    kRTC\_TimeInvalidFlag = RTC\_SR\_TIF\_MASK,  
    kRTC\_TimeOverflowFlag = RTC\_SR\_TOF\_MASK,  
    kRTC\_AlarmFlag = RTC\_SR\_TAF\_MASK }  
*List of RTC flags.*

## Functions

- static void `RTC_Reset` (RTC\_Type \*base)  
*Performs a software reset on the RTC module.*

## Driver version

- #define `FSL_RTC_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
*Version 2.0.0.*

## Initialization and deinitialization

- void `RTC_Init` (RTC\_Type \*base, const `rtc_config_t` \*config)  
*Ungates the RTC clock and configures the peripheral for basic operation.*
- static void `RTC_Deinit` (RTC\_Type \*base)  
*Stop the timer and gate the RTC clock.*
- void `RTC_GetDefaultConfig` (`rtc_config_t` \*config)  
*Fill in the RTC config struct with the default settings.*

## Current Time & Alarm

- status\_t **RTC\_SetDatetime** (RTC\_Type \*base, const rtc\_datetime\_t \*datetime)  
*Sets the RTC date and time according to the given time structure.*
- void **RTC\_GetDatetime** (RTC\_Type \*base, rtc\_datetime\_t \*datetime)  
*Gets the RTC time and stores it in the given time structure.*
- status\_t **RTC\_SetAlarm** (RTC\_Type \*base, const rtc\_datetime\_t \*alarmTime)  
*Sets the RTC alarm time.*
- void **RTC\_GetAlarm** (RTC\_Type \*base, rtc\_datetime\_t \*datetime)  
*Returns the RTC alarm time.*

## Interrupt Interface

- static void **RTC\_EnableInterrupts** (RTC\_Type \*base, uint32\_t mask)  
*Enables the selected RTC interrupts.*
- static void **RTC\_DisableInterrupts** (RTC\_Type \*base, uint32\_t mask)  
*Disables the selected RTC interrupts.*
- static uint32\_t **RTC\_GetEnabledInterrupts** (RTC\_Type \*base)  
*Gets the enabled RTC interrupts.*

## Status Interface

- static uint32\_t **RTC\_GetStatusFlags** (RTC\_Type \*base)  
*Gets the RTC status flags.*
- void **RTC\_ClearStatusFlags** (RTC\_Type \*base, uint32\_t mask)  
*Clears the RTC status flags.*

## Timer Start and Stop

- static void **RTC\_StartTimer** (RTC\_Type \*base)  
*Starts the RTC time counter.*
- static void **RTC\_StopTimer** (RTC\_Type \*base)  
*Stops the RTC time counter.*

## 29.4 Data Structure Documentation

### 29.4.1 struct rtc\_datetime\_t

#### Data Fields

- uint16\_t **year**  
*Range from 1970 to 2099.*
- uint8\_t **month**  
*Range from 1 to 12.*
- uint8\_t **day**  
*Range from 1 to 31 (depending on month).*
- uint8\_t **hour**  
*Range from 0 to 23.*
- uint8\_t **minute**  
*Range from 0 to 59.*

## Enumeration Type Documentation

- `uint8_t second`

*Range from 0 to 59.*

### 29.4.1.0.0.11 Field Documentation

#### 29.4.1.0.0.11.1 `uint16_t rtc_datetime_t::year`

#### 29.4.1.0.0.11.2 `uint8_t rtc_datetime_t::month`

#### 29.4.1.0.0.11.3 `uint8_t rtc_datetime_t::day`

#### 29.4.1.0.0.11.4 `uint8_t rtc_datetime_t::hour`

#### 29.4.1.0.0.11.5 `uint8_t rtc_datetime_t::minute`

#### 29.4.1.0.0.11.6 `uint8_t rtc_datetime_t::second`

## 29.4.2 `struct rtc_config_t`

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the [RTC\\_GetDefaultConfig\(\)](#) function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

## Data Fields

- `bool wakeupSelect`  
*true: Wakeup pin outputs the 32 KHz clock; false:Wakeup pin used to wakeup the chip*
- `bool updateMode`  
*true: Registers can be written even when locked under certain conditions, false: No writes allowed when registers are locked*
- `bool supervisorAccess`  
*true: Non-supervisor accesses are allowed; false: Non-supervisor accesses are not supported*
- `uint32_t compensationInterval`  
*Compensation interval that is written to the CIR field in RTC TCR Register.*
- `uint32_t compensationTime`  
*Compensation time that is written to the TCR field in RTC TCR Register.*

## 29.5 Enumeration Type Documentation

### 29.5.1 `enum rtc_interrupt_enable_t`

Enumerator

`kRTC_TimeInvalidInterruptEnable` Time invalid interrupt.

`kRTC_TimeOverflowInterruptEnable` Time overflow interrupt.

`kRTC_AlarmInterruptEnable` Alarm interrupt.

***kRTC\_SecondsInterruptEnable*** Seconds interrupt.

## 29.5.2 enum rtc\_status\_flags\_t

Enumerator

***kRTC\_TimeInvalidFlag*** Time invalid flag.

***kRTC\_TimeOverflowFlag*** Time overflow flag.

***kRTC\_AlarmFlag*** Alarm flag.

## 29.6 Function Documentation

### 29.6.1 void RTC\_Init ( RTC\_Type \* *base*, const rtc\_config\_t \* *config* )

This function will issue a software reset if the timer invalid flag is set.

Note

This API should be called at the beginning of the application using the RTC driver.

Parameters

<i>base</i>	RTC peripheral base address
<i>config</i>	Pointer to user's RTC config structure.

### 29.6.2 static void RTC\_Deinit ( RTC\_Type \* *base* ) [inline], [static]

Parameters

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

### 29.6.3 void RTC\_GetDefaultConfig ( rtc\_config\_t \* *config* )

The default values are:

```
*     config->wakeupSelect = false;
*     config->updateMode = false;
*     config->supervisorAccess = false;
*     config->compensationInterval = 0;
*     config->compensationTime = 0;
*
```

## Function Documentation

Parameters

<i>config</i>	Pointer to user's RTC config structure.
---------------	---

### 29.6.4 **status\_t RTC\_SetDatetime ( RTC\_Type \* *base*, const rtc\_datetime\_t \* *datetime* )**

The RTC counter must be stopped prior to calling this function as writes to the RTC seconds register will fail if the RTC counter is running.

Parameters

<i>base</i>	RTC 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 RTC  
kStatus\_InvalidArgument: Error because the datetime format is incorrect

### 29.6.5 **void RTC\_GetDatetime ( RTC\_Type \* *base*, rtc\_datetime\_t \* *datetime* )**

Parameters

<i>base</i>	RTC peripheral base address
<i>datetime</i>	Pointer to structure where the date and time details are stored.

### 29.6.6 **status\_t RTC\_SetAlarm ( RTC\_Type \* *base*, const rtc\_datetime\_t \* *alarmTime* )**

The function checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

Parameters

<i>base</i>	RTC peripheral base address
<i>alarmTime</i>	Pointer to structure where the alarm time is stored.

Returns

kStatus\_Success: success in setting the RTC alarm  
kStatus\_InvalidArgument: Error because the alarm datetime format is incorrect  
kStatus\_Fail: Error because the alarm time has already passed

### 29.6.7 void RTC\_GetAlarm ( RTC\_Type \* *base*, rtc\_datetime\_t \* *datetime* )

Parameters

<i>base</i>	RTC peripheral base address
<i>datetime</i>	Pointer to structure where the alarm date and time details are stored.

### 29.6.8 static void RTC\_EnableInterrupts ( RTC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	RTC peripheral base address
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">rtc_interrupt_enable_t</a>

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

Parameters

<i>base</i>	RTC peripheral base address
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">rtc_interrupt_enable_t</a>

### 29.6.10 static uint32\_t RTC\_GetEnabledInterrupts ( RTC\_Type \* *base* ) [inline], [static]

## Function Documentation

Parameters

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

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [rtc\\_interrupt\\_enable\\_t](#)

### 29.6.11 static uint32\_t RTC\_GetStatusFlags ( RTC\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

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

### 29.6.12 void RTC\_ClearStatusFlags ( RTC\_Type \* *base*, uint32\_t *mask* )

Parameters

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

### 29.6.13 static void RTC\_StartTimer ( RTC\_Type \* *base* ) [inline], [static]

After calling this function, the timer counter increments once a second provided SR[TOF] or SR[TIF] are not set.

Parameters

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

### 29.6.14 static void RTC\_StopTimer( RTC\_Type \* *base* ) [inline], [static]

RTC's seconds register can be written to only when the timer is stopped.

Parameters

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

### 29.6.15 static void RTC\_Reset( RTC\_Type \* *base* ) [inline], [static]

This resets all RTC registers except for the SWR bit and the RTC\_WAR and RTC\_RAR registers. The SWR bit is cleared by software explicitly clearing it.

Parameters

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

## Function Documentation

# Chapter 30

## SIM: System Integration Module Driver

### 30.1 Overview

The KSDK provides a peripheral driver for the System Integration Module (SIM) of Kinetis 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`)  
*Get the unique identification register value.*
- static void `SIM_SetFlashMode` (`uint8_t mode`)  
*Set the flash enable mode.*

### Driver version

- #define `FSL_SIM_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 0)`)  
*Driver version 2.0.0.*

### 30.2 Data Structure Documentation

#### 30.2.1 struct `sim_uid_t`

### Data Fields

- `uint32_t MH`  
*UIDMH.*
- `uint32_t ML`  
*UIDML.*
- `uint32_t L`  
*UIDL.*

## Function Documentation

### 30.2.1.0.0.12 Field Documentation

30.2.1.0.0.12.1 `uint32_t sim_uid_t::MH`

30.2.1.0.0.12.2 `uint32_t sim_uid_t::ML`

30.2.1.0.0.12.3 `uint32_t sim_uid_t::L`

## 30.3 Enumeration Type Documentation

### 30.3.1 `enum _sim_flash_mode`

Enumerator

`kSIM_FlashDisableInWait` Disable flash in wait mode.

`kSIM_FlashDisable` Disable flash in normal mode.

## 30.4 Function Documentation

### 30.4.1 `void SIM_GetUniqueId ( sim_uid_t * uid )`

Parameters

<code>uid</code>	Pointer to the structure to save the UID value.
------------------	---

### 30.4.2 `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 31

## Smart Card

### 31.1 Overview

The Kinetis SDK provides Peripheral drivers for the UART-ISO7816 and EMVSIM modules of Kinetis devices.

Smart Card driver provides the necessary functions to access and control integrated circuit cards. The driver controls communication modules (UART/EMVSIM) and handles special ICC sequences, such as the activation/deactivation (using EMVSIM IP or external interface chip). The Smart Card driver consists of two IPs (SmartCard\_Uart and SmartCard\_EmvSim drivers) and three PHY drivers (smartcard\_phy\_emvsim, smartcard\_phy\_tda8035 and smartcard\_phy\_gpio drivers). These drivers can be combined, which means that the Smart Card driver wraps one IP (transmission) and one PHY (interface) driver.

The driver provides asynchronous functions to communicate with the Integrated Circuit Card (ICC). The driver contains RTOS adaptation layers which use semaphores as synchronization objects of synchronous transfers. The RTOS driver support also provides protection for multithreading.

### 31.2 SmartCard Driver Initialization

The Smart Card Driver is initialized by calling the [SMARTCARD\\_Init\(\)](#) and [SMARTCARD\\_PHY\\_Init\(\)](#) functions. The Smart Card Driver initialization configuration structure requires these settings:

- Smart Card voltage class
- Smart Card Interface options such as the RST, IRQ, CLK pins, and so on.

The driver also supports user callbacks for assertion/de-assertion Smart Card events and transfer finish event. This feature is useful to detect the card presence or for handling transfer events i.e., in RTOS. The user should initialize the Smart Card driver, which consist of IP and PHY drivers.

### 31.3 SmartCard Call diagram

Because the call diagram is complex, the detailed use of the Smart Card driver is not described in this section. For details about using the Smart Card driver, see the Smart Card driver example which describes a simple use case.

### 31.4 PHY driver

The Smart Card interface driver is initialized by calling the function [SMARTCARD\\_PHY\\_Init\(\)](#). During the initialization phase, Smart Card clock is configured and all hardware pins for IC handling are configured.

```
/*!
```

## PHY driver

### Modules

- Smart Card EMVSIM Driver
- Smart Card FreeRTOS Driver
- Smart Card PHY EMVSIM Driver
- Smart Card PHY GPIO Driver
- Smart Card PHY TDA8035 Driver
- Smart Card UART Driver
- Smart Card µCOS/II Driver
- Smart Card µCOS/III Driver

### Data Structures

- struct `smartcard_card_params_t`  
*Defines card-specific parameters for Smart card driver. [More...](#)*
- struct `smartcard_timers_state_t`  
*Smart card Defines the state of the EMV timers in the Smart card driver. [More...](#)*
- struct `smartcard_interface_config_t`  
*Defines user specified configuration of Smart card interface. [More...](#)*
- struct `smartcard_xfer_t`  
*Defines user transfer structure used to initialize transfer. [More...](#)*
- struct `smartcard_context_t`  
*Runtime state of the Smart card driver. [More...](#)*

### Macros

- #define `SMARTCARD_INIT_DELAY_CLOCK_CYCLES` (42000u)  
*Smart card global define which specify number of clock cycles until initial 'TS' character has to be received.*
- #define `SMARTCARD_EMV_ATR_DURATION_ETU` (20150u)  
*Smart card global define which specify number of clock cycles during which ATR string has to be received.*
- #define `SMARTCARD_TS_DIRECT_CONVENTION` (0x3Bu)  
*Smart card specification initial TS character definition of direct convention.*
- #define `SMARTCARD_TS_INVERSE_CONVENTION` (0x3Fu)  
*Smart card specification initial TS character definition of inverse convention.*

### Typedefs

- typedef void(\* `smartcard_interface_callback_t`)(void \*smartcardContext, void \*param)  
*Smart card interface interrupt callback function type.*
- typedef void(\* `smartcard_transfer_callback_t`)(void \*smartcardContext, void \*param)  
*Smart card transfer interrupt callback function type.*
- typedef void(\* `smartcard_time_delay_t`)(uint32\_t miliseconds)  
*Time Delay function used to passive waiting using RTOS [ms].*

### Enumerations

- enum `smartcard_status_t` {  
  kStatus\_SMARTCARD\_Success = MAKE\_STATUS(kStatusGroup\_SMARTCARD, 0),  
  kStatus\_SMARTCARD\_TxBusy = MAKE\_STATUS(kStatusGroup\_SMARTCARD, 1),  
  kStatus\_SMARTCARD\_RxBusy = MAKE\_STATUS(kStatusGroup\_SMARTCARD, 2),  
  kStatus\_SMARTCARD\_NoTransferInProgress = MAKE\_STATUS(kStatusGroup\_SMARTCAR-

```
D, 3),
kStatus_SMARTCARD_Timeout = MAKE_STATUS(kStatusGroup_SMARTCARD, 4),
kStatus_SMARTCARD_Initialized,
kStatus_SMARTCARD_PhyInitialized,
kStatus_SMARTCARD_CardNotActivated = MAKE_STATUS(kStatusGroup_SMARTCARD, 7),
kStatus_SMARTCARD_InvalidInput,
kStatus_SMARTCARD_OtherError = MAKE_STATUS(kStatusGroup_SMARTCARD, 9) }
```

*Smart card Error codes.*

- enum **smartcard\_control\_t**  
*Control codes for the Smart card protocol timers and misc.*
- enum **smartcard\_card\_voltage\_class\_t**  
*Defines Smart card interface voltage class values.*
- enum **smartcard\_transfer\_state\_t**  
*Defines Smart card I/O transfer states.*
- enum **smartcard\_reset\_type\_t**  
*Defines Smart card reset types.*
- enum **smartcard\_transport\_type\_t**  
*Defines Smart card transport protocol types.*
- enum **smartcard\_parity\_type\_t**  
*Defines Smart card data parity types.*
- enum **smartcard\_card\_convention\_t**  
*Defines data Convention format.*
- enum **smartcard\_interface\_control\_t**  
*Defines Smart card interface IC control types.*
- enum **smartcard\_direction\_t**  
*Defines transfer direction.*

## Driver version

- #define **FSL\_SMARTCARD\_DRIVER\_VERSION** (MAKE\_VERSION(2, 1, 1))  
*Smart card driver version 2.1.1.*

## 31.5 Data Structure Documentation

### 31.5.1 struct smartcard\_card\_params\_t

#### Data Fields

- uint16\_t **Fi**  
*4 bits Fi - clock rate conversion integer*
- uint8\_t **fMax**  
*Maximum Smart card frequency in MHz.*
- uint8\_t **WI**  
*8 bits WI - work wait time integer*
- uint8\_t **Di**  
*4 bits DI - baud rate divisor*
- uint8\_t **BWI**  
*4 bits BWI - block wait time integer*
- uint8\_t **CWI**

## Data Structure Documentation

- `uint8_t BGI`  
*4 bits CWI - character wait time integer*
- `uint8_t GTN`  
*4 bits BGI - block guard time integer*
- `uint8_t IFSC`  
*8 bits GTN - extended guard time integer*
- `uint8_t modeNegotiable`  
*Indicates IFSC value of the card.*
- `uint8_t currentD`  
*Indicates if the card acts in negotiable or a specific mode.*
- `uint8_t status`  
*Indicates smart card status.*
- `bool t0Indicated`  
*Indicates ff T=0 indicated in TD1 byte.*
- `bool t1Indicated`  
*Indicates if T=1 indicated in TD2 byte.*
- `bool atrComplete`  
*Indicates whether the ATR received from the card was complete or not.*
- `bool atrValid`  
*Indicates whether the ATR received from the card was valid or not.*
- `bool present`  
*Indicates if a smart card is present.*
- `bool active`  
*Indicates if the smart card is activated.*
- `bool faulty`  
*Indicates whether smart card/interface is faulty.*
- `smartcard_card_convention_t convention`  
*Card convention, kSMARTCARD\_DirectConvention for direct convention, kSMARTCARD\_InverseConvention for inverse convention.*

### 31.5.1.0.0.13 Field Documentation

#### 31.5.1.0.0.13.1 `uint8_t smartcard_card_params_t::modeNegotiable`

### 31.5.2 `struct smartcard_timers_state_t`

#### Data Fields

- `volatile bool adtExpired`  
*Indicates whether ADT timer expired.*
- `volatile bool wwtExpired`  
*Indicates whether WWT timer expired.*
- `volatile bool cwtExpired`  
*Indicates whether CWT timer expired.*
- `volatile bool bwtExpired`  
*Indicates whether BWT timer expired.*
- `volatile bool initCharTimerExpired`  
*Indicates whether reception timer for initialization character (TS) after the RST has expired*

### 31.5.3 struct smartcard\_interface\_config\_t

#### Data Fields

- `uint32_t smartCardClock`  
*Smart card interface clock [Hz].*
- `uint32_t clockToResetDelay`  
*Indicates clock to RST apply delay [smart card clock cycles].*
- `uint8_t clockModule`  
*Smart card clock module number.*
- `uint8_t clockModuleChannel`  
*Smart card clock module channel number.*
- `uint8_t clockModuleSourceClock`  
*Smart card clock module source clock [e.g., BusClk].*
- `smartcard_card_voltage_class_t vcc`  
*Smart card voltage class.*
- `uint8_t controlPort`  
*Smart card PHY control port instance.*
- `uint8_t controlPin`  
*Smart card PHY control pin instance.*
- `uint8_t irqPort`  
*Smart card PHY Interrupt port instance.*
- `uint8_t irqPin`  
*Smart card PHY Interrupt pin instance.*
- `uint8_t resetPort`  
*Smart card reset port instance.*
- `uint8_t resetPin`  
*Smart card reset pin instance.*
- `uint8_t vsel0Port`  
*Smart card PHY Vsel0 control port instance.*
- `uint8_t vsel0Pin`  
*Smart card PHY Vsel0 control pin instance.*
- `uint8_t vsel1Port`  
*Smart card PHY Vsel1 control port instance.*
- `uint8_t vsel1Pin`  
*Smart card PHY Vsel1 control pin instance.*
- `uint8_t dataPort`  
*Smart card PHY data port instance.*
- `uint8_t dataPin`  
*Smart card PHY data pin instance.*
- `uint8_t dataPinMux`  
*Smart card PHY data pin mux option.*
- `uint8_t tsTimerId`  
*Numerical identifier of the External HW timer for Initial character detection.*

## Data Structure Documentation

### 31.5.4 struct smartcard\_xfer\_t

#### Data Fields

- **smartcard\_direction\_t direction**  
*Direction of communication.*
- **uint8\_t \* buff**  
*The buffer of data.*
- **size\_t size**  
*The number of transferred units.*

#### 31.5.4.0.0.14 Field Documentation

##### 31.5.4.0.0.14.1 smartcard\_direction\_t smartcard\_xfer\_t::direction

(RX/TX)

##### 31.5.4.0.0.14.2 uint8\_t\* smartcard\_xfer\_t::buff

##### 31.5.4.0.0.14.3 size\_t smartcard\_xfer\_t::size

### 31.5.5 struct smartcard\_context\_t

#### Data Fields

- **void \* base**  
*Smart card module base address.*
- **smartcard\_direction\_t direction**  
*Direction of communication.*
- **uint8\_t \* xBuff**  
*The buffer of data being transferred.*
- **volatile size\_t xSize**  
*The number of bytes to be transferred.*
- **volatile bool xIsBusy**  
*True if there is an active transfer.*
- **uint8\_t txFifoEntryCount**  
*Number of data word entries in transmit FIFO.*
- **smartcard\_interface\_callback\_t interfaceCallback**  
*Callback to invoke after interface IC raised interrupt.*
- **smartcard\_transfer\_callback\_t transferCallback**  
*Callback to invoke after transfer event occur.*
- **void \* interfaceCallbackParam**  
*Interface callback parameter pointer.*
- **void \* transferCallbackParam**  
*Transfer callback parameter pointer.*
- **smartcard\_time\_delay\_t timeDelay**  
*Function which handles time delay defined by user or RTOS.*
- **smartcard\_reset\_type\_t resetType**  
*Indicates whether a Cold reset or Warm reset was requested.*

- **smartcard\_transport\_type\_t tType**  
*Indicates current transfer protocol (T0 or T1)*
- volatile **smartcard\_transfer\_state\_t transferState**  
*Indicates the current transfer state.*
- **smartcard\_timers\_state\_t timersState**  
*Indicates the state of different protocol timers used in driver.*
- **smartcard\_card\_params\_t cardParams**  
*Smart card parameters(ATR and current) and interface slots states(ATR and current)*
- uint8\_t **IFSD**  
*Indicates the terminal IFSD.*
- **smartcard\_parity\_type\_t parity**  
*Indicates current parity even/odd.*
- volatile bool **rxtCrossed**  
*Indicates whether RXT thresholds has been crossed.*
- volatile bool **txtCrossed**  
*Indicates whether TXT thresholds has been crossed.*
- volatile bool **wtxRequested**  
*Indicates whether WTX has been requested or not.*
- volatile bool **parityError**  
*Indicates whether a parity error has been detected.*
- uint8\_t **statusBytes [2]**  
*Used to store Status bytes SW1, SW2 of the last executed card command response.*
- **smartcard\_interface\_config\_t interfaceConfig**  
*Smart card interface configuration structure.*

### 31.5.5.0.0.15 Field Documentation

#### 31.5.5.0.0.15.1 smartcard\_direction\_t smartcard\_context\_t::direction

(RX/TX)

#### 31.5.5.0.0.15.2 uint8\_t\* smartcard\_context\_t::xBuff

#### 31.5.5.0.0.15.3 volatile size\_t smartcard\_context\_t::xSize

#### 31.5.5.0.0.15.4 volatile bool smartcard\_context\_t::xIsBusy

#### 31.5.5.0.0.15.5 uint8\_t smartcard\_context\_t::txFifoEntryCount

#### 31.5.5.0.0.15.6 smartcard\_interface\_callback\_t smartcard\_context\_t::interfaceCallback

#### 31.5.5.0.0.15.7 smartcard\_transfer\_callback\_t smartcard\_context\_t::transferCallback

#### 31.5.5.0.0.15.8 void\* smartcard\_context\_t::interfaceCallbackParam

#### 31.5.5.0.0.15.9 void\* smartcard\_context\_t::transferCallbackParam

#### 31.5.5.0.0.15.10 smartcard\_time\_delay\_t smartcard\_context\_t::timeDelay

#### 31.5.5.0.0.15.11 smartcard\_reset\_type\_t smartcard\_context\_t::resetType

## Enumeration Type Documentation

### 31.6 Enumeration Type Documentation

#### 31.6.1 enum smartcard\_status\_t

Enumerator

- kStatus\_SMARTCARD\_Success* Transfer ends successfully.
- kStatus\_SMARTCARD\_TxBusy* Transmit in progress.
- kStatus\_SMARTCARD\_RxBusy* Receiving in progress.
- kStatus\_SMARTCARD\_NoTransferInProgress* No transfer in progress.
- kStatus\_SMARTCARD\_Timeout* Transfer ends with time-out.
- kStatus\_SMARTCARD\_Initialized* Smart card driver is already initialized.
- kStatus\_SMARTCARD\_PhyInitialized* Smart card PHY drive is already initialized.
- kStatus\_SMARTCARD\_CardNotActivated* Smart card is not activated.
- kStatus\_SMARTCARD\_InvalidInput* Function called with invalid input arguments.
- kStatus\_SMARTCARD\_OtherError* Some other error occur.

#### 31.6.2 enum smartcard\_control\_t

#### 31.6.3 enum smartcard\_direction\_t

## 31.7 Smart Card PHY TDA8035 Driver

### 31.7.1 Overview

The Smart Card interface TDA8035 driver handles the external interface chip TDA8035 which supports all necessary functions to control the ICC. These functions involve PHY pin initialization, ICC voltage selection and activation, ICC clock generation, ICC card detection, and activation/deactivation sequences.

## Macros

- #define **SMARTCARD\_ATR\_DURATION\_ADJUSTMENT** (360u)  
*Smart card definition which specifies adjustment number of clock cycles during which ATR string has to be received.*
- #define **SMARTCARD\_INIT\_DELAY\_CLOCK\_CYCLES\_ADJUSTMENT** (4200u)  
*Smart card definition which specifies adjustment number of clock cycles until initial 'TS' character has to be received.*
- #define **SMARTCARD\_TDA8035\_STATUS\_PRES** (0x01u)  
*Masks for TDA8035 status register.*
- #define **SMARTCARD\_TDA8035\_STATUS\_ACTIVE** (0x02u)  
*Smart card phy TDA8035 Smart card active status.*
- #define **SMARTCARD\_TDA8035\_STATUS\_FAULTY** (0x04u)  
*Smart card phy TDA8035 Smart card faulty status.*
- #define **SMARTCARD\_TDA8035\_STATUS\_CARD\_REMOVED** (0x08u)  
*Smart card phy TDA8035 Smart card removed status.*
- #define **SMARTCARD\_TDA8035\_STATUS\_CARD\_DEACTIVATED** (0x10u)  
*Smart card phy TDA8035 Smart card deactivated status.*

## Functions

- void **SMARTCARD\_PHY\_TDA8035\_GetDefaultConfig** (smartcard\_interface\_config\_t \*config)  
*Fills in config structure with default values.*
- status\_t **SMARTCARD\_PHY\_TDA8035\_Init** (void \*base, smartcard\_interface\_config\_t const \*config, uint32\_t srcClock\_Hz)  
*Initializes a Smart card interface instance for operation.*
- void **SMARTCARD\_PHY\_TDA8035\_Deinit** (void \*base, smartcard\_interface\_config\_t \*config)  
*De-initializes a Smart card interface.*
- status\_t **SMARTCARD\_PHY\_TDA8035\_Activate** (void \*base, smartcard\_context\_t \*context, smartcard\_reset\_type\_t resetType)  
*Activates the Smart card IC.*
- status\_t **SMARTCARD\_PHY\_TDA8035\_Deactivate** (void \*base, smartcard\_context\_t \*context)  
*De-activates the Smart card IC.*
- status\_t **SMARTCARD\_PHY\_TDA8035\_Control** (void \*base, smartcard\_context\_t \*context, smartcard\_interface\_control\_t control, uint32\_t param)  
*Controls Smart card interface IC.*
- void **SMARTCARD\_PHY\_TDA8035\_IRQHandler** (void \*base, smartcard\_context\_t \*context)  
*Smart card interface IC IRQ ISR.*

### 31.7.2 Macro Definition Documentation

#### 31.7.2.1 #define SMARTCARD\_TDA8035\_STATUS\_PRES (0x01u)

Smart card phy TDA8035 Smart card present status

### 31.7.3 Function Documentation

#### 31.7.3.1 void SMARTCARD\_PHY\_TDA8035\_GetDefaultConfig ( smartcard\_interface\_config\_t \* config )

Parameters

<i>config</i>	The Smart card user configuration structure which contains configuration structure of type <a href="#">smartcard_interface_config_t</a> . Function fill in members: clockToResetDelay = 42000, vcc = kSmartcardVoltageClassB3_3V, with default values.
---------------	--

#### 31.7.3.2 status\_t SMARTCARD\_PHY\_TDA8035\_Init ( void \* base, smartcard\_interface\_config\_t const \* config, uint32\_t srcClock\_Hz )

Parameters

<i>base</i>	The Smart card peripheral base address.
<i>config</i>	The user configuration structure of type <a href="#">smartcard_interface_config_t</a> . The user can call to fill out configuration structure function <a href="#">SMARTCARD_PHY_TDA8035_GetDefaultConfig()</a> .
<i>srcClock_Hz</i>	Smart card clock generation module source clock.

Return values

<i>kStatus_SMARTCARD_Success</i>	or <i>kStatus_SMARTCARD_OtherError</i> in case of error.
----------------------------------	--

#### 31.7.3.3 void SMARTCARD\_PHY\_TDA8035\_Deinit ( void \* base, smartcard\_interface\_config\_t \* config )

Stops Smart card clock and disable VCC.

Parameters

<i>base</i>	The Smart card peripheral module base address.
<i>config</i>	The user configuration structure of type <a href="#">smartcard_interface_config_t</a> .

### 31.7.3.4 status\_t SMARTCARD\_PHY\_TDA8035\_Activate ( void \* *base*, smartcard\_context\_t \* *context*, smartcard\_reset\_type\_t *resetType* )

Parameters

<i>base</i>	The Smart card peripheral module base address.
<i>context</i>	A pointer to a Smart card driver context structure.
<i>resetType</i>	type of reset to be performed, possible values = kSmartcardColdReset, kSmartcardWarmReset

Return values

<i>kStatus_SMARTCARD_Success</i>	or kStatus_SMARTCARD_OtherError in case of error.
----------------------------------	---

### 31.7.3.5 status\_t SMARTCARD\_PHY\_TDA8035\_Deactivate ( void \* *base*, smartcard\_context\_t \* *context* )

Parameters

<i>base</i>	The Smart card peripheral module base address.
<i>context</i>	A pointer to a Smart card driver context structure.

Return values

<i>kStatus_SMARTCARD_Success</i>	or kStatus_SMARTCARD_OtherError in case of error.
----------------------------------	---

### 31.7.3.6 status\_t SMARTCARD\_PHY\_TDA8035\_Control ( void \* *base*, smartcard\_context\_t \* *context*, smartcard\_interface\_control\_t *control*, uint32\_t *param* )

## Smart Card PHY TDA8035 Driver

Parameters

<i>base</i>	The Smart card peripheral module base address.
<i>context</i>	A pointer to a Smart card driver context structure.
<i>control</i>	A interface command type.
<i>param</i>	Integer value specific to control type

Return values

<i>kStatus_SMARTCARD_-Success</i>	or kStatus_SMARTCARD_OtherError in case of error.
-----------------------------------	---

### 31.7.3.7 void SMARTCARD\_PHY\_TDA8035\_IRQHandler ( void \* *base*, smartcard\_context\_t \* *context* )

Parameters

<i>base</i>	The Smart card peripheral module base address.
<i>context</i>	The Smart card context pointer.

# Chapter 32

## Smart Card PHY TDA8035 Driver

### 32.1 Overview

The Smart Card interface TDA8035 driver handles the external interface chip TDA8035 which supports all necessary functions to control the ICC. These functions involve PHY pin initialization, ICC voltage selection and activation, ICC clock generation, ICC card detection, and activation/deactivation sequences.

### Macros

- #define **SMARTCARD\_ATR\_DURATION\_ADJUSTMENT** (360u)  
*Smart card definition which specifies adjustment number of clock cycles during which ATR string has to be received.*
- #define **SMARTCARD\_INIT\_DELAY\_CLOCK\_CYCLES\_ADJUSTMENT** (4200u)  
*Smart card definition which specifies adjustment number of clock cycles until initial 'TS' character has to be received.*
- #define **SMARTCARD\_NCN8025\_STATUS\_PRES** (0x01u)  
*Masks for NCN8025 status register.*
- #define **SMARTCARD\_NCN8025\_STATUS\_ACTIVE** (0x02u)  
*Smart card phy NCN8025 Smart card active status.*
- #define **SMARTCARD\_NCN8025\_STATUS\_FAULTY** (0x04u)  
*Smart card phy NCN8025 Smart card faulty status.*
- #define **SMARTCARD\_NCN8025\_STATUS\_CARD\_REMOVED** (0x08u)  
*Smart card phy NCN8025 Smart card removed status.*
- #define **SMARTCARD\_NCN8025\_STATUS\_CARD\_DEACTIVATED** (0x10u)  
*Smart card phy NCN8025 Smart card deactivated status.*

### Functions

- void **SMARTCARD\_PHY\_NCN8025\_GetDefaultConfig** (smartcard\_interface\_config\_t \*config)  
*Fills in config structure with default values.*
- status\_t **SMARTCARD\_PHY\_NCN8025\_Init** (void \*base, smartcard\_interface\_config\_t const \*config, uint32\_t srcClock\_Hz)  
*Initializes a Smart card interface instance for operation.*
- void **SMARTCARD\_PHY\_NCN8025\_Deinit** (void \*base, smartcard\_interface\_config\_t \*config)  
*De-initializes an Smart card interface.*
- status\_t **SMARTCARD\_PHY\_NCN8025\_Activate** (void \*base, smartcard\_context\_t \*context, smartcard\_reset\_type\_t resetType)  
*Activates the Smart card IC.*
- status\_t **SMARTCARD\_PHY\_NCN8025\_Deactivate** (void \*base, smartcard\_context\_t \*context)  
*De-activates the Smart card IC.*
- status\_t **SMARTCARD\_PHY\_NCN8025\_Control** (void \*base, smartcard\_context\_t \*context, smartcard\_interface\_control\_t control, uint32\_t param)  
*Controls Smart card interface IC.*
- void **SMARTCARD\_PHY\_NCN8025\_IRQHandler** (void \*base, smartcard\_context\_t \*context)

## Function Documentation

Smart card interface IC IRQ ISR.

### 32.2 Macro Definition Documentation

#### 32.2.1 #define SMARTCARD\_NCN8025\_STATUS\_PRES (0x01u)

Smart card phy NCN8025 Smart card present status

### 32.3 Function Documentation

#### 32.3.1 void SMARTCARD\_PHY\_NCN8025\_GetDefaultConfig ( smartcard\_interface\_config\_t \* config )

Parameters

<i>config</i>	The Smart card user configuration structure which contains configuration structure of type <a href="#">smartcard_interface_config_t</a> . Function fill in members: clockToResetDelay = 42000, vcc = kSmartcardVoltageClassB3_3V, with default values.
---------------	--

#### 32.3.2 status\_t SMARTCARD\_PHY\_NCN8025\_Init ( void \* base, smartcard\_interface\_config\_t const \* config, uint32\_t srcClock\_Hz )

Parameters

<i>base</i>	The Smart card peripheral base address.
<i>config</i>	The user configuration structure of type <a href="#">smartcard_interface_config_t</a> . The user can call to fill out configuration structure function <a href="#">SMARTCARD_PHY_NCN8025_GetDefaultConfig()</a> .
<i>srcClock_Hz</i>	Smart card clock generation module source clock.

Return values

<i>kStatus_SMARTCARD_Success</i>	or <i>kStatus_SMARTCARD_OtherError</i> in case of error.
----------------------------------	--

#### 32.3.3 void SMARTCARD\_PHY\_NCN8025\_Deinit ( void \* base, smartcard\_interface\_config\_t \* config )

Stops Smart card clock and disable VCC.

Parameters

<i>base</i>	The Smart card peripheral module base address.
<i>config</i>	The user configuration structure of type <a href="#">smartcard_interface_config_t</a> .

### 32.3.4 **status\_t SMARTCARD\_PHY\_NCN8025\_Activate ( void \* *base*, smartcard\_context\_t \* *context*, smartcard\_reset\_type\_t *resetType* )**

Parameters

<i>base</i>	The Smart card peripheral module base address.
<i>context</i>	A pointer to a Smart card driver context structure.
<i>resetType</i>	type of reset to be performed, possible values = kSmartcardColdReset, kSmartcard-WarmReset

Return values

<i>kStatus_SMARTCARD_-Success</i>	or kStatus_SMARTCARD_OtherError in case of error.
-----------------------------------	---

### 32.3.5 **status\_t SMARTCARD\_PHY\_NCN8025\_Deactivate ( void \* *base*, smartcard\_context\_t \* *context* )**

Parameters

<i>base</i>	The Smart card peripheral module base address.
<i>context</i>	A pointer to a Smart card driver context structure.

Return values

<i>kStatus_SMARTCARD_-Success</i>	or kStatus_SMARTCARD_OtherError in case of error.
-----------------------------------	---

### 32.3.6 **status\_t SMARTCARD\_PHY\_NCN8025\_Control ( void \* *base*, smartcard\_context\_t \* *context*, smartcard\_interface\_control\_t *control*, uint32\_t *param* )**

## Function Documentation

Parameters

<i>base</i>	The Smart card peripheral module base address.
<i>context</i>	A pointer to a Smart card driver context structure.
<i>control</i>	A interface command type.
<i>param</i>	Integer value specific to control type

Return values

<i>kStatus_SMARTCARD_-Success</i>	or kStatus_SMARTCARD_OtherError in case of error.
-----------------------------------	---

### 32.3.7 void SMARTCARD\_PHY\_NCN8025\_IRQHandler ( void \* *base*, smartcard\_context\_t \* *context* )

Parameters

<i>base</i>	The Smart card peripheral module base address.
<i>context</i>	The Smart card context pointer.

## 32.4 Smart Card PHY EMVSIM Driver

### 32.4.1 Overview

The Smart Card interface EMVSIM driver handles the EMVSIM peripheral, which covers all necessary functions to control the ICC. These functions are ICC clock setup, ICC voltage turning on/off, ICC card detection, activation/deactivation, and ICC reset sequences. The EMVSIM peripheral covers all features of interface ICC chips.

### Macros

- #define **SMARTCARD\_ATR\_DURATION\_ADJUSTMENT** (360u)  
*Smart card define which specify adjustment number of clock cycles during which ATR string has to be received.*
- #define **SMARTCARD\_INIT\_DELAY\_CLOCK\_CYCLES\_ADJUSTMENT** (4200u)  
*Smart card define which specify adjustment number of clock cycles until initial 'TS' character has to be received.*

### Functions

- void **SMARTCARD\_PHY\_EMVSIM\_GetDefaultConfig** (smartcard\_interface\_config\_t \*config)  
*Fill in smartcardInterfaceConfig structure with default values.*
- status\_t **SMARTCARD\_PHY\_EMVSIM\_Init** (EMVSIM\_Type \*base, const smartcard\_interface\_config\_t \*config, uint32\_t srcClock\_Hz)  
*Configures a Smart card interface for operation.*
- void **SMARTCARD\_PHY\_EMVSIM\_Deinit** (EMVSIM\_Type \*base, const smartcard\_interface\_config\_t \*config)  
*De-initializes a Smart card interface.*
- status\_t **SMARTCARD\_PHY\_EMVSIM\_Activate** (EMVSIM\_Type \*base, smartcard\_context\_t \*context, smartcard\_reset\_type\_t resetType)  
*Activates the smart card IC.*
- status\_t **SMARTCARD\_PHY\_EMVSIM\_Deactivate** (EMVSIM\_Type \*base, smartcard\_context\_t \*context)  
*De-activates the smart card IC.*
- status\_t **SMARTCARD\_PHY\_EMVSIM\_Control** (EMVSIM\_Type \*base, smartcard\_context\_t \*context, smartcard\_interface\_control\_t control, uint32\_t param)  
*Controls Smart card interface IC.*

### 32.4.2 Function Documentation

#### 32.4.2.1 void SMARTCARD\_PHY\_EMVSIM\_GetDefaultConfig ( smartcard\_interface\_config\_t \* config )

## Smart Card PHY EMVSIM Driver

Parameters

<i>config</i>	The user configuration structure of type <a href="#">smartcard_interface_config_t</a> . Function fill in members: clockToResetDelay = 42000, vcc = kSmartcardVoltageClassB3_3V, with default values.
---------------	--

### 32.4.2.2 **status\_t SMARTCARD\_PHY\_EMVSIM\_Init ( *EMVSIM\_Type \* base*, *const smartcard\_interface\_config\_t \* config*, *uint32\_t srcClock\_Hz* )**

Parameters

<i>base</i>	The Smart card peripheral module base address.
<i>config</i>	The user configuration structure of type <a href="#">smartcard_interface_config_t</a> . The user is responsible to fill out the members of this structure and to pass the pointer of this structure into this function or call SMARTCARD_PHY_EMVSIMInitUserConfig. Default to fill out structure with default values.
<i>srcClock_Hz</i>	Smart card clock generation module source clock.

Return values

<i>kStatus_SMARTCARD_-Success</i>	or kStatus_SMARTCARD_OtherError in case of error.
-----------------------------------	---

### 32.4.2.3 **void SMARTCARD\_PHY\_EMVSIM\_Deinit ( *EMVSIM\_Type \* base*, *const smartcard\_interface\_config\_t \* config* )**

Stops Smart card clock and disable VCC.

Parameters

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

### 32.4.2.4 **status\_t SMARTCARD\_PHY\_EMVSIM\_Activate ( *EMVSIM\_Type \* base*, *smartcard\_context\_t \* context*, *smartcard\_reset\_type\_t resetType* )**

Parameters

<i>base</i>	The EMVSIM peripheral base address.
<i>context</i>	A pointer to a Smart card driver context structure.
<i>resetType</i>	type of reset to be performed, possible values = kSmartcardColdReset, kSmartcard-WarmReset

Return values

<i>kStatus_SMARTCARD_Success</i>	or kStatus_SMARTCARD_OtherError in case of error.
----------------------------------	---

### 32.4.2.5 status\_t SMARTCARD\_PHY\_EMVSIM\_Deactivate ( EMVSIM\_Type \* *base*, smartcard\_context\_t \* *context* )

Parameters

<i>base</i>	The EMVSIM peripheral base address.
<i>context</i>	A pointer to a Smart card driver context structure.

Return values

<i>kStatus_SMARTCARD_Success</i>	or kStatus_SMARTCARD_OtherError in case of error.
----------------------------------	---

### 32.4.2.6 status\_t SMARTCARD\_PHY\_EMVSIM\_Control ( EMVSIM\_Type \* *base*, smartcard\_context\_t \* *context*, smartcard\_interface\_control\_t *control*, uint32\_t *param* )

Parameters

<i>base</i>	The EMVSIM peripheral base address.
<i>context</i>	A pointer to a Smart card driver context structure.
<i>control</i>	A interface command type.

## Smart Card PHY EMVSIM Driver

<i>param</i>	Integer value specific to control type
--------------	--

Return values

<i>kStatus_SMARTCARD_-Success</i>	or kStatus_SMARTCARD_OtherError in case of error.
-----------------------------------	---

## 32.5 Smart Card PHY GPIO Driver

### 32.5.1 Overview

The Smart Card interface GPIO driver handles the GPIO and FTM/TPM peripheral for clock generation, which covers all necessary functions to control the ICC. These functions are ICC clock setup, ICC voltage turning on/off, activation/deactivation, and ICC reset sequences. This driver doesn't support the ICC pin short circuit protection and an emergency deactivation.

## Macros

- #define **SMARTCARD\_ATR\_DURATION\_ADJUSTMENT** (360u)  
*Smart card define which specify adjustment number of clock cycles during which ATR string has to be received.*
- #define **SMARTCARD\_INIT\_DELAY\_CLOCK\_CYCLES\_ADJUSTMENT** (4200u)  
*Smart card define which specify adjustment number of clock cycles until initial 'TS' character has to be received.*

## Functions

- void **SMARTCARD\_PHY\_GPIO\_GetDefaultConfig** (**smartcard\_interface\_config\_t** \*config)  
*Fill in config structure with default values.*
- status\_t **SMARTCARD\_PHY\_GPIO\_Init** (UART\_Type \*base, **smartcard\_interface\_config\_t** const \*config, uint32\_t srcClock\_Hz)  
*Initializes a Smart card interface instance for operation.*
- void **SMARTCARD\_PHY\_GPIO\_Deinit** (UART\_Type \*base, **smartcard\_interface\_config\_t** \*config)  
*De-initializes a Smart card interface.*
- status\_t **SMARTCARD\_PHY\_GPIO\_Activate** (UART\_Type \*base, **smartcard\_context\_t** \*context, **smartcard\_reset\_type\_t** resetType)  
*Activates the Smart card IC.*
- status\_t **SMARTCARD\_PHY\_GPIO\_Deactivate** (UART\_Type \*base, **smartcard\_context\_t** \*context)  
*De-activates the Smart card IC.*
- status\_t **SMARTCARD\_PHY\_GPIO\_Control** (UART\_Type \*base, **smartcard\_context\_t** \*context, **smartcard\_interface\_control\_t** control, uint32\_t param)  
*Controls Smart card interface IC.*

### 32.5.2 Function Documentation

#### 32.5.2.1 void **SMARTCARD\_PHY\_GPIO\_GetDefaultConfig** ( **smartcard\_interface\_config\_t** \* config )

## Smart Card PHY GPIO Driver

Parameters

<i>config</i>	The Smart card user configuration structure which contains configuration structure of type <a href="#">smartcard_interface_config_t</a> . Function fill in members: <code>clockToResetDelay</code> = 42000, <code>vcc</code> = <code>kSmartcardVoltageClassB3_3V</code> , with default values.
---------------	--

### 32.5.2.2 **status\_t SMARTCARD\_PHY\_GPIO\_Init ( UART\_Type \* *base*, smartcard\_interface\_config\_t const \* *config*, uint32\_t *srcClock\_Hz* )**

Parameters

<i>base</i>	The Smart card peripheral module base address.
<i>config</i>	The user configuration structure of type <a href="#">smartcard_interface_config_t</a> . The user can call to fill out configuration structure function <a href="#">SMARTCARD_PHY_GPIO_GetDefaultConfig()</a> .
<i>srcClock_Hz</i>	Smart card clock generation module source clock.

Return values

<i>kStatus_SMARTCARD_Success</i>	or <code>kStatus_SMARTCARD_OtherError</code> in case of error.
----------------------------------	--

### 32.5.2.3 **void SMARTCARD\_PHY\_GPIO\_Deinit ( UART\_Type \* *base*, smartcard\_interface\_config\_t \* *config* )**

Stops Smart card clock and disable VCC.

Parameters

<i>base</i>	The Smart card peripheral module base address.
<i>config</i>	The user configuration structure of type <a href="#">smartcard_interface_config_t</a> .

### 32.5.2.4 **status\_t SMARTCARD\_PHY\_GPIO\_Activate ( UART\_Type \* *base*, smartcard\_context\_t \* *context*, smartcard\_reset\_type\_t *resetType* )**

Parameters

<i>base</i>	The Smart card peripheral module base address.
<i>context</i>	A pointer to a Smart card driver context structure.
<i>resetType</i>	type of reset to be performed, possible values = kSmartcardColdReset, kSmartcard-WarmReset

Return values

<i>kStatus_SMARTCARD_Success</i>	or kStatus_SMARTCARD_OtherError in case of error.
----------------------------------	---

### 32.5.2.5 status\_t SMARTCARD\_PHY\_GPIO\_Deactivate ( UART\_Type \* *base*, smartcard\_context\_t \* *context* )

Parameters

<i>base</i>	The Smart card peripheral module base address.
<i>context</i>	A pointer to a Smart card driver context structure.

Return values

<i>kStatus_SMARTCARD_Success</i>	or kStatus_SMARTCARD_OtherError in case of error.
----------------------------------	---

### 32.5.2.6 status\_t SMARTCARD\_PHY\_GPIO\_Control ( UART\_Type \* *base*, smartcard\_context\_t \* *context*, smartcard\_interface\_control\_t *control*, uint32\_t *param* )

Parameters

<i>base</i>	The Smart card peripheral module base address.
<i>context</i>	A pointer to a Smart card driver context structure.
<i>control</i>	A interface command type.

## Smart Card PHY GPIO Driver

<i>param</i>	Integer value specific to control type
--------------	--

Return values

<i>kStatus_SMARTCARD_-Success</i>	or kStatus_SMARTCARD_OtherError in case of error.
-----------------------------------	---

## 32.6 Smart Card UART Driver

### 32.6.1 Overview

The Smart Card UART driver uses a standard UART peripheral which supports the ISO-7816 standard. The driver supports transmission functionality in the CPU mode. The driver also supports non-blocking (asynchronous) type of data transfers. The blocking (synchronous) transfer is supported only by the RTOS adaptation layer.

## Macros

- #define **SMARTCARD\_EMV\_RX\_NACK\_THRESHOLD** (5u)  
*EMV RX NACK interrupt generation threshold.*
- #define **SMARTCARD\_EMV\_TX\_NACK\_THRESHOLD** (3u)  
*EMV TX NACK interrupt generation threshold.*
- #define **SMARTCARD\_EMV\_RX\_TO\_TX\_GUARD\_TIME\_T0** (0x0u)  
*EMV TX & RX GUART TIME default value.*

## Functions

- void **SMARTCARD\_UART\_GetDefaultConfig** (**smartcard\_card\_params\_t** \*cardParams)  
*Fill in smartcard\_card\_params structure with default values according EMV 4.3 specification.*
- status\_t **SMARTCARD\_UART\_Init** (UART\_Type \*base, **smartcard\_context\_t** \*context, uint32\_t srcClock\_Hz)  
*Initializes an UART peripheral for smart card/ISO-7816 operation.*
- void **SMARTCARD\_UART\_Deinit** (UART\_Type \*base)  
*This function disables the UART interrupts, disables the transmitter and receiver, and flushes the FIFOs (for modules that support FIFOs) and gates UART clock in SIM.*
- int32\_t **SMARTCARD\_UART\_GetTransferRemainingBytes** (UART\_Type \*base, **smartcard\_context\_t** \*context)  
*Returns whether the previous UART transfer has finished.*
- status\_t **SMARTCARD\_UART\_AbortTransfer** (UART\_Type \*base, **smartcard\_context\_t** \*context)  
*Terminates an asynchronous UART transfer early.*
- status\_t **SMARTCARD\_UART\_TransferNonBlocking** (UART\_Type \*base, **smartcard\_context\_t** \*context, **smartcard\_xfer\_t** \*xfer)  
*Transfer data using interrupts.*
- status\_t **SMARTCARD\_UART\_Control** (UART\_Type \*base, **smartcard\_context\_t** \*context, **smartcard\_control\_t** control, uint32\_t param)  
*Controls UART module as per different user request.*
- void **SMARTCARD\_UART\_IRQHandler** (UART\_Type \*base, **smartcard\_context\_t** \*context)  
*Interrupt handler for UART.*
- void **SMARTCARD\_UART\_ErrIRQHandler** (UART\_Type \*base, **smartcard\_context\_t** \*context)  
*Error Interrupt handler for UART.*
- void **SMARTCARD\_UART\_TSExpiryCallback** (UART\_Type \*base, **smartcard\_context\_t** \*context)  
*Handles initial TS character timer time-out event.*

## 32.6.2 Function Documentation

32.6.2.1 **void SMARTCARD\_UART\_GetDefaultConfig ( smartcard\_card\_params\_t \*  
*cardParams* )**

Parameters

<i>cardParams</i>	The configuration structure of type <code>smartcard_interface_config_t</code> . Function fill in members: Fi = 372; Di = 1; currentD = 1; WI = 0x0A; GTN = 0x00; with default values.
-------------------	---

### 32.6.2.2 `status_t SMARTCARD_UART_Init ( UART_Type * base, smartcard_context_t * context, uint32_t srcClock_Hz )`

This function Un-gate UART clock, initializes the module to EMV default settings, configures the IRQ, enables the module-level interrupt to the core and initialize driver context.

Parameters

<i>base</i>	The UART peripheral base address.
<i>context</i>	A pointer to a smart card driver context structure.
<i>srcClock_Hz</i>	Smart card clock generation module source clock.

Returns

An error code or `kStatus_SMARTCARD_Success`.

### 32.6.2.3 `void SMARTCARD_UART_Deinit ( UART_Type * base )`

Parameters

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

### 32.6.2.4 `int32_t SMARTCARD_UART_GetTransferRemainingBytes ( UART_Type * base, smartcard_context_t * context )`

When performing an async transfer, call this function to ascertain the context of the current transfer: in progress (or busy) or complete (success). If the transfer is still in progress, the user can obtain the number of words that have not been transferred, by reading `xSize` of smart card context structure.

Parameters

## Smart Card UART Driver

<i>base</i>	The UART peripheral base address.
<i>context</i>	A pointer to a smart card driver context structure.

Returns

The number of bytes not transferred.

### 32.6.2.5 status\_t SMARTCARD\_UART\_AbortTransfer ( **UART\_Type \* base,** **smartcard\_context\_t \* context** )

During an async UART transfer, the user can terminate the transfer early if the transfer is still in progress.

Parameters

<i>base</i>	The UART peripheral base address.
<i>context</i>	A pointer to a smart card driver context structure.

Return values

<i>kStatus_SMARTCARD_-Success</i>	The transfer abort was successful.
<i>kStatus_SMARTCARD_-NoTransmitInProgress</i>	No transmission is currently in progress.

### 32.6.2.6 status\_t SMARTCARD\_UART\_TransferNonBlocking ( **UART\_Type \* base,** **smartcard\_context\_t \* context, smartcard\_xfer\_t \* xfer** )

A non-blocking (also known as asynchronous) function means that the function returns immediately after initiating the transfer function. The application has to get the transfer status to see when the transfer is complete. In other words, after calling non-blocking (asynchronous) transfer function, the application must get the transfer status to check if transmit is completed or not.

Parameters

<i>base</i>	The UART peripheral base address.
<i>context</i>	A pointer to a smart card driver context structure.

<i>xfer</i>	A pointer to smart card transfer structure where are linked buffers and sizes.
-------------	--

Returns

An error code or kStatus\_SMARTCARD\_Success.

### 32.6.2.7 **status\_t SMARTCARD\_UART\_Control ( UART\_Type \* *base*, smartcard\_context\_t \* *context*, smartcard\_control\_t *control*, uint32\_t *param* )**

Parameters

<i>base</i>	The UART peripheral base address.
<i>context</i>	A pointer to a smart card driver context structure.
<i>control</i>	Smart card command type.
<i>param</i>	Integer value of specific to control command.

return An kStatus\_SMARTCARD\_OtherError in case of error return kStatus\_SMARTCARD\_Success in success

### 32.6.2.8 **void SMARTCARD\_UART\_IRQHandler ( UART\_Type \* *base*, smartcard\_context\_t \* *context* )**

This handler uses the buffers stored in the [smartcard\\_context\\_t](#) structures to transfer data. Smart card driver requires this function to call when UART interrupt occurs.

Parameters

<i>base</i>	The UART peripheral base address.
<i>context</i>	A pointer to a smart card driver context structure.

### 32.6.2.9 **void SMARTCARD\_UART\_ErrIRQHandler ( UART\_Type \* *base*, smartcard\_context\_t \* *context* )**

This function handles error conditions during transfer.

Parameters

## Smart Card UART Driver

<i>base</i>	The UART peripheral base address.
<i>context</i>	A pointer to a smart card driver context structure.

**32.6.2.10 void SMARTCARD\_UART\_TSExpiryCallback ( **UART\_Type** \* *base*, **smartcard\_context\_t** \* *context* )**

Parameters

<i>base</i>	The UART peripheral base address.
<i>context</i>	A pointer to a smart card driver context structure.

## 32.7 Smart Card EMVSIM Driver

### 32.7.1 Overview

The SmartCard EMVSIM driver covers the transmission functionality in the CPU mode. The driver supports non-blocking (asynchronous) type of data transfers. The blocking (synchronous) transfer is supported only by the RTOS adaptation layer.

### Macros

- #define **SMARTCARD\_EMV\_RX\_NACK\_THRESHOLD** (5u)  
*EMV RX NACK interrupt generation threshold.*
- #define **SMARTCARD\_EMV\_TX\_NACK\_THRESHOLD** (5u)  
*EMV TX NACK interrupt generation threshold.*
- #define **SMARTCARD\_WWT\_ADJUSTMENT** (160u)  
*Smart card Word Wait Timer adjustment value.*
- #define **SMARTCARD\_CWT\_ADJUSTMENT** (3u)  
*Smart card Character Wait Timer adjustment value.*

### Enumerations

- enum **emvsim\_gpc\_clock\_select\_t** {
   
  kEMVSIM\_GPCClockDisable = 0u,
   
  kEMVSIM\_GPCCardClock = 1u,
   
  kEMVSIM\_GPCRxClock = 2u,
   
  kEMVSIM\_GPCTxClock = 3u }
   
*General Purpose Counter clock selections.*
- enum **emvsim\_presence\_detect\_edge\_t** {
   
  kEMVSIM\_DetectOnFallingEdge = 0u,
   
  kEMVSIM\_DetectOnRisingEdge = 1u }
   
*EMVSIM card presence detection edge control.*
- enum **emvsim\_presence\_detect\_status\_t** {
   
  kEMVSIM\_DetectPinIsLow = 0u,
   
  kEMVSIM\_DetectPinIsHigh = 1u }
   
*EMVSIM card presence detection status.*

### Smart card EMVSIM Driver

- void **SMARTCARD\_EMVSIM\_GetDefaultConfig** (smartcard\_card\_params\_t \*cardParams)
   
*Fill in smartcard\_card\_params structure with default values according EMV 4.3 specification.*
- status\_t **SMARTCARD\_EMVSIM\_Init** (EMVSIM\_Type \*base, smartcard\_context\_t \*context, uint32\_t srcClock\_Hz)
   
*Initializes an EMVSIM peripheral for smart card/ISO-7816 operation.*
- void **SMARTCARD\_EMVSIM\_Deinit** (EMVSIM\_Type \*base)

## Smart Card EMVSIM Driver

*This function disables the EMVSIM interrupts, disables the transmitter and receiver, flushes the FIFOs and gates EMVSIM clock in SIM.*

- int32\_t **SMARTCARD\_EMVSIM\_GetTransferRemainingBytes** (EMVSIM\_Type \*base, smartcard\_context\_t \*context)  
*Returns whether the previous EMVSIM transfer has finished.*
- status\_t **SMARTCARD\_EMVSIM\_AbortTransfer** (EMVSIM\_Type \*base, smartcard\_context\_t \*context)  
*Terminates an asynchronous EMVSIM transfer early.*
- status\_t **SMARTCARD\_EMVSIM\_TransferNonBlocking** (EMVSIM\_Type \*base, smartcard\_context\_t \*context, smartcard\_xfer\_t \*xfer)  
*Transfer data using interrupts.*
- status\_t **SMARTCARD\_EMVSIM\_Control** (EMVSIM\_Type \*base, smartcard\_context\_t \*context, smartcard\_control\_t control, uint32\_t param)  
*Controls EMVSIM module as per different user request.*
- void **SMARTCARD\_EMVSIM\_IRQHandler** (EMVSIM\_Type \*base, smartcard\_context\_t \*context)  
*Handles EMVSIM module interrupts.*

### 32.7.2 Enumeration Type Documentation

#### 32.7.2.1 enum emvsim\_gpc\_clock\_select\_t

Enumerator

**kEMVSIM\_GPCClockDisable** disabled  
**kEMVSIM\_GPCCardClock** card clock  
**kEMVSIM\_GPCRxClock** receive clock  
**kEMVSIM\_GPCTxClock** transmit ETU clock

#### 32.7.2.2 enum emvsim\_presence\_detect\_edge\_t

Enumerator

**kEMVSIM\_DetectOnFallingEdge** presence detect on falling edge  
**kEMVSIM\_DetectOnRisingEdge** presence detect on rising edge

#### 32.7.2.3 enum emvsim\_presence\_detect\_status\_t

Enumerator

**kEMVSIM\_DetectPinIsLow** presence detect pin is logic low  
**kEMVSIM\_DetectPinIsHigh** presence detect pin is logic high

### 32.7.3 Function Documentation

32.7.3.1 **void SMARTCARD\_EMVSIM\_GetDefaultConfig ( smartcard\_card\_params\_t \*  
*cardParams* )**

## Smart Card EMVSIM Driver

Parameters

<i>cardParams</i>	The configuration structure of type <a href="#">smartcard_interface_config_t</a> . Function fill in members: Fi = 372; Di = 1; currentD = 1; WI = 0x0A; GTN = 0x00; with default values.
-------------------	--

### 32.7.3.2 **status\_t SMARTCARD\_EMVSIM\_Init ( *EMVSIM\_Type* \* *base*, *smartcard\_context\_t* \* *context*, *uint32\_t srcClock\_Hz* )**

This function Un-gate EMVSIM clock, initializes the module to EMV default settings, configures the IRQ, enables the module-level interrupt to the core and initialize driver context.

Parameters

<i>base</i>	The EMVSIM peripheral base address.
<i>context</i>	A pointer to a smart card driver context structure.
<i>srcClock_Hz</i>	Smart card clock generation module source clock.

Returns

An error code or kStatus\_SMARTCARD\_Success.

### 32.7.3.3 **void SMARTCARD\_EMVSIM\_Deinit ( *EMVSIM\_Type* \* *base* )**

Parameters

<i>base</i>	The EMVSIM module base address.
-------------	---------------------------------

### 32.7.3.4 **int32\_t SMARTCARD\_EMVSIM\_GetTransferRemainingBytes ( *EMVSIM\_Type* \* *base*, *smartcard\_context\_t* \* *context* )**

When performing an async transfer, call this function to ascertain the context of the current transfer: in progress (or busy) or complete (success). If the transfer is still in progress, the user can obtain the number of words that have not been transferred.

Parameters

<i>base</i>	The EMVSIM module base address.
<i>context</i>	A pointer to a smart card driver context structure.

Returns

The number of bytes not transferred.

### 32.7.3.5 status\_t SMARTCARD\_EMVSIM\_AbortTransfer ( EMVSIM\_Type \* *base*, smartcard\_context\_t \* *context* )

During an async EMVSIM transfer, the user can terminate the transfer early if the transfer is still in progress.

Parameters

<i>base</i>	The EMVSIM peripheral address.
<i>context</i>	A pointer to a smart card driver context structure.

Return values

<i>kStatus_SMARTCARD_Success</i>	The transmit abort was successful.
<i>kStatus_SMARTCARD_NoTransmitInProgress</i>	No transmission is currently in progress.

### 32.7.3.6 status\_t SMARTCARD\_EMVSIM\_TransferNonBlocking ( EMVSIM\_Type \* *base*, smartcard\_context\_t \* *context*, smartcard\_xfer\_t \* *xfer* )

A non-blocking (also known as asynchronous) function means that the function returns immediately after initiating the transfer function. The application has to get the transfer status to see when the transfer is complete. In other words, after calling non-blocking (asynchronous) transfer function, the application must get the transfer status to check if transmit is completed or not.

Parameters

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

## Smart Card EMVSIM Driver

<i>context</i>	A pointer to a smart card driver context structure.
<i>xfer</i>	A pointer to smart card transfer structure where are linked buffers and sizes.

Returns

An error code or kStatus\_SMARTCARD\_Success.

### 32.7.3.7 **status\_t SMARTCARD\_EMVSIM\_Control ( EMVSIM\_Type \* *base*, smartcard\_context\_t \* *context*, smartcard\_control\_t *control*, uint32\_t *param* )**

Parameters

<i>base</i>	The EMVSIM peripheral base address.
<i>context</i>	A pointer to a smart card driver context structure.
<i>control</i>	Control type
<i>param</i>	Integer value of specific to control command.

return kStatus\_SMARTCARD\_Success in success. return kStatus\_SMARTCARD\_OtherError in case of error.

### 32.7.3.8 **void SMARTCARD\_EMVSIM\_IRQHandler ( EMVSIM\_Type \* *base*, smartcard\_context\_t \* *context* )**

Parameters

<i>base</i>	The EMVSIM peripheral base address.
<i>context</i>	A pointer to a smart card driver context structure.

## 32.8 Smart Card FreeRTOS Driver

### 32.8.1 Overview

#### Data Structures

- struct `rtos_smartcard_context_t`  
*Runtime RTOS smart card driver context.* [More...](#)

#### Macros

- #define `RTOS_SMARTCARD_COMPLETE` 0x1u  
*smart card RTOS transfer complete flag*
- #define `RTOS_SMARTCARD_TIMEOUT` 0x2u  
*smart card RTOS transfer time-out flag*
- #define `SMARTCARD_Control`(base, context, control, param) `SMARTCARD_UART_Control`(base, context, control, 0)  
*Common smart card driver API defines.*
- #define `SMARTCARD_Transfer`(base, context, xfer) `SMARTCARD_UART_TransferNonBlocking`(base, context, xfer)  
*Common smart card API macro.*
- #define `SMARTCARD_Init`(base, context, sourceClockHz) `SMARTCARD_UART_Init`(base, context, sourceClockHz)  
*Common smart card API macro.*
- #define `SMARTCARD_Deinit`(base) `SMARTCARD_UART_Deinit`(base)  
*Common smart card API macro.*
- #define `SMARTCARD_GetTransferRemainingBytes`(base, context) `SMARTCARD_UART_GetTransferRemainingBytes`(base, context)  
*Common smart card API macro.*
- #define `SMARTCARD_GetDefaultConfig`(cardParams) `SMARTCARD_UART_GetDefaultConfig`(cardParams)  
*Common smart card API macro.*
- #define `SMARTCARD_PHY_Activate`(base, context, resetType) `SMARTCARD_PHY_GPIO_Activate`(base, context, resetType)  
*Common smart card API macro.*
- #define `SMARTCARD_PHY_Deactivate`(base, context) `SMARTCARD_PHY_GPIO_Deactivate`(base, context)  
*Common smart card API macro.*
- #define `SMARTCARD_PHY_Control`(base, context, control, param) `SMARTCARD_PHY_GPIO_Control`(base, context, control, param)  
*Common smart card API macro.*
- #define `SMARTCARD_PHY_Init`(base, config, sourceClockHz) `SMARTCARD_PHY_GPIO_Init`(base, config, sourceClockHz)  
*Common smart card API macro.*
- #define `SMARTCARD_PHY_Deinit`(base, config) `SMARTCARD_PHY_GPIO_Deinit`(base, config)  
*Common smart card API macro.*

## Smart Card FreeRTOS Driver

- #define **SMARTCARD\_PHY\_GetDefaultConfig**(config) **SMARTCARD\_PHY\_GPIO\_GetDefaultConfig**(config)  
*Common smart card API macro.*

## Functions

- int **SMARTCARD\_RTOS\_Init** (void \*base, rtos\_smartcard\_context\_t \*ctx, uint32\_t sourceClock-Hz)  
*Initializes a smart card (EMVSIM/UART) peripheral for smart card/ISO-7816 operation.*
- int **SMARTCARD\_RTOS\_Deinit** (rtos\_smartcard\_context\_t \*ctx)  
*This function disables the smart card (EMVSIM/UART) interrupts, disables the transmitter and receiver, and flushes the FIFOs (for modules that support FIFOs) and gates smart card clock in SIM.*
- int **SMARTCARD\_RTOS\_Transfer** (rtos\_smartcard\_context\_t \*ctx, smartcard\_xfer\_t \*xfer)  
*Transfers data using interrupts.*
- int **SMARTCARD\_RTOS\_WaitForXevent** (rtos\_smartcard\_context\_t \*ctx)  
*Waits until the transfer is finished.*
- int **SMARTCARD\_RTOS\_Control** (rtos\_smartcard\_context\_t \*ctx, smartcard\_control\_t control, uint32\_t param)  
*Controls the smart card module as per different user request.*
- int **SMARTCARD\_RTOS\_PHY\_Control** (rtos\_smartcard\_context\_t \*ctx, smartcard\_interface-control\_t control, uint32\_t param)  
*Controls the smart card module as per different user request.*
- int **SMARTCARD\_RTOS\_PHY\_Activate** (rtos\_smartcard\_context\_t \*ctx, smartcard\_reset\_type\_t resetType)  
*Activates the smart card interface.*
- int **SMARTCARD\_RTOS\_PHY\_Deactivate** (rtos\_smartcard\_context\_t \*ctx)  
*Deactivates the smart card interface.*

### 32.8.2 Data Structure Documentation

#### 32.8.2.1 struct rtos\_smartcard\_context\_t

Runtime RTOS Smart card driver context.

#### Data Fields

- SemaphoreHandle\_t **x\_sem**  
*RTOS unique access assurance object.*
- EventGroupHandle\_t **x\_event**  
*RTOS synchronization object.*
- smartcard\_context\_t **x\_context**  
*transactional layer state*
- OS\_EVENT \* **x\_sem**  
*RTOS unique access assurance object.*
- OS\_FLAG\_GRP \* **x\_event**  
*RTOS synchronization object.*

- OS\_SEM `x_sem`  
*RTOS unique access assurance object.*
- OS\_FLAG\_GRP `x_event`  
*RTOS synchronization object.*

### 32.8.2.1.0.16 Field Documentation

#### 32.8.2.1.0.16.1 `smartcard_context_t rtos_smartcard_context_t::x_context`

Transactional layer state.

### 32.8.3 Macro Definition Documentation

#### 32.8.3.1 `#define SMARTCARD_Control( base, context, control, param ) SMARTCARD_UART_Control(base, context, control, 0)`

Common smart card API macro

### 32.8.4 Function Documentation

#### 32.8.4.1 `int SMARTCARD_RTOS_Init ( void * base, rtos_smartcard_context_t * ctx, uint32_t sourceClockHz )`

Also initialize smart card PHY interface .

This function ungates the smart card clock, initializes the module to EMV default settings, configures the IRQ state structure, and enables the module-level interrupt to the core. Initialize RTOS synchronization objects and context.

Parameters

<code>base</code>	The smart card peripheral base address.
<code>ctx</code>	The smart card RTOS structure.
<code>sourceClockHz</code>	smart card clock generation module source clock.

Returns

An zero in Success or error code.

#### 32.8.4.2 `int SMARTCARD_RTOS_Deinit ( rtos_smartcard_context_t * ctx )`

Deactivates also smart card PHY interface, stops smart card clocks. Free all synchronization objects allocated in RTOS smart card context.

## Smart Card FreeRTOS Driver

Parameters

<i>ctx</i>	The smart card RTOS state.
------------	----------------------------

Returns

An zero in Success or error code.

### **32.8.4.3 int SMARTCARD\_RTOS\_Transfer ( rtos\_smartcard\_context\_t \* *ctx*, smartcard\_xfer\_t \* *xfer* )**

A blocking (also known as synchronous) function means that the function returns after the transfer is done. User can cancel this transfer by calling function AbortTransfer.

Parameters

<i>ctx</i>	A pointer to the RTOS smart card driver context.
<i>xfer</i>	smart card transfer structure.

Returns

An zero in Success or error code.

### **32.8.4.4 int SMARTCARD\_RTOS\_WaitForXevent ( rtos\_smartcard\_context\_t \* *ctx* )**

Task waits on a transfer finish event. Don't initialize the transfer. Instead, wait for transfer callback. Can be used while waiting on initial TS character.

Parameters

<i>ctx</i>	A pointer to the RTOS smart card driver context.
------------	--

Returns

A zero in Success or error code.

### **32.8.4.5 int SMARTCARD\_RTOS\_Control ( rtos\_smartcard\_context\_t \* *ctx*, smartcard\_control\_t *control*, uint32\_t *param* )**

Parameters

<i>ctx</i>	The smart card RTOS context pointer.
<i>control</i>	Control type
<i>param</i>	Integer value of specific to control command.

Returns

An zero in Success or error code.

### 32.8.4.6 int SMARTCARD\_RTOS\_PHY\_Control ( *rtos\_smartcard\_context\_t \* ctx*, *smartcard\_interface\_control\_t control*, *uint32\_t param* )

Parameters

<i>ctx</i>	The smart card RTOS context pointer.
<i>control</i>	Control type
<i>param</i>	Integer value of specific to control command.

Returns

An zero in Success or error code.

### 32.8.4.7 int SMARTCARD\_RTOS\_PHY\_Activate ( *rtos\_smartcard\_context\_t \* ctx*, *smartcard\_reset\_type\_t resetType* )

Parameters

<i>ctx</i>	The smart card RTOS driver context structure.
<i>resetType</i>	type of reset to be performed, possible values = kSmartcardColdReset, kSmartcardWarmReset

Returns

An zero in Success or error code.

### 32.8.4.8 int SMARTCARD\_RTOS\_PHY\_Deactivate ( *rtos\_smartcard\_context\_t \* ctx* )

## Smart Card FreeRTOS Driver

### Parameters

<i>ctx</i>	The smart card RTOS driver context structure.
------------	---

### Returns

An zero in Success or error code.

## 32.9 Smart Card µCOS/II Driver

### 32.9.1 Overview

#### Data Structures

- struct `rtos_smartcard_context_t`  
*Runtime RTOS smart card driver context.* [More...](#)

#### Macros

- #define `RTOS_SMARTCARD_COMPLETE` 0x1u  
*Smart card RTOS transfer complete flag.*
- #define `RTOS_SMARTCARD_TIMEOUT` 0x2u  
*Smart card RTOS transfer time-out flag.*
- #define `SMARTCARD_Control`(base, context, control, param) `SMARTCARD_UART_Control`(base, context, control, 0)  
*Common Smart card driver API defines.*
- #define `SMARTCARD_Transfer`(base, context, xfer) `SMARTCARD_UART_TransferNonBlocking`(base, context, xfer)  
*Common Smart card API macro.*
- #define `SMARTCARD_Init`(base, context, sourceClockHz) `SMARTCARD_UART_Init`(base, context, sourceClockHz)  
*Common Smart card API macro.*
- #define `SMARTCARD_Deinit`(base) `SMARTCARD_UART_Deinit`(base)  
*Common Smart card API macro.*
- #define `SMARTCARD_GetTransferRemainingBytes`(base, context) `SMARTCARD_UART_GetTransferRemainingBytes`(base, context)  
*Common Smart card API macro.*
- #define `SMARTCARD_GetDefaultConfig`(cardParams) `SMARTCARD_UART_GetDefaultConfig`(cardParams)  
*Common Smart card API macro.*
- #define `SMARTCARD_PHY_Activate`(base, context, resetType) `SMARTCARD_PHY_GPIO_Activate`(base, context, resetType)  
*Common Smart card API macro.*
- #define `SMARTCARD_PHY_Deactivate`(base, context) `SMARTCARD_PHY_GPIO_Deactivate`(base, context)  
*Common Smart card API macro.*
- #define `SMARTCARD_PHY_Control`(base, context, control, param) `SMARTCARD_PHY_GPIO_Control`(base, context, control, param)  
*Common Smart card API macro.*
- #define `SMARTCARD_PHY_Init`(base, config, sourceClockHz) `SMARTCARD_PHY_GPIO_Init`(base, config, sourceClockHz)  
*Common Smart card API macro.*
- #define `SMARTCARD_PHY_Deinit`(base, config) `SMARTCARD_PHY_GPIO_Deinit`(base, config)  
*Common Smart card API macro.*

## Smart Card µCOS/II Driver

- #define **SMARTCARD\_PHY\_GetDefaultConfig**(config) **SMARTCARD\_PHY\_GPIO\_GetDefaultConfig**(config)  
*Common Smart card API macro.*

## Functions

- int **SMARTCARD\_RTOS\_Init** (void \*base, rtos\_smartcard\_context\_t \*ctx, uint32\_t sourceClock-Hz)  
*Initializes an Smart card (EMVSIM/UART) peripheral for Smart card/ISO-7816 operation.*
- int **SMARTCARD\_RTOS\_Deinit** (rtos\_smartcard\_context\_t \*ctx)  
*This function disables the Smart card (EMVSIM/UART) interrupts, disables the transmitter and receiver, and flushes the FIFOs (for modules that support FIFOs) and gates Smart card clock in SIM.*
- int **SMARTCARD\_RTOS\_Transfer** (rtos\_smartcard\_context\_t \*ctx, smartcard\_xfer\_t \*xfer)  
*Transfers data using interrupts.*
- int **SMARTCARD\_RTOS\_WaitForXevent** (rtos\_smartcard\_context\_t \*ctx)  
*Waits until transfer is finished.*
- int **SMARTCARD\_RTOS\_Control** (rtos\_smartcard\_context\_t \*ctx, smartcard\_control\_t control, uint32\_t param)  
*Controls Smart card module as per different user request.*
- int **SMARTCARD\_RTOS\_PHY\_Control** (rtos\_smartcard\_context\_t \*ctx, smartcard\_interface-control\_t control, uint32\_t param)  
*Controls the Smart card module as per different user request.*
- int **SMARTCARD\_RTOS\_PHY\_Activate** (rtos\_smartcard\_context\_t \*ctx, smartcard\_reset\_type\_t resetType)  
*Activates the Smart card interface.*
- int **SMARTCARD\_RTOS\_PHY\_Deactivate** (rtos\_smartcard\_context\_t \*ctx)  
*Deactivates the Smart card interface.*

### 32.9.2 Data Structure Documentation

#### 32.9.2.1 struct rtos\_smartcard\_context\_t

Runtime RTOS Smart card driver context.

#### Data Fields

- SemaphoreHandle\_t **x\_sem**  
*RTOS unique access assurance object.*
- EventGroupHandle\_t **x\_event**  
*RTOS synchronization object.*
- smartcard\_context\_t **x\_context**  
*transactional layer state*
- OS\_EVENT \* **x\_sem**  
*RTOS unique access assurance object.*
- OS\_FLAG\_GRP \* **x\_event**  
*RTOS synchronization object.*

- OS\_SEM `x_sem`  
*RTOS unique access assurance object.*
- OS\_FLAG\_GRP `x_event`  
*RTOS synchronization object.*

### 32.9.2.1.0.17 Field Documentation

#### 32.9.2.1.0.17.1 `smartcard_context_t rtos_smartcard_context_t::x_context`

Transactional layer state.

### 32.9.3 Macro Definition Documentation

#### 32.9.3.1 `#define SMARTCARD_Control( base, context, control, param ) SMARTCARD_UART_Control(base, context, control, 0)`

Common Smart card API macro

### 32.9.4 Function Documentation

#### 32.9.4.1 `int SMARTCARD_RTOS_Init ( void * base, rtos_smartcard_context_t * ctx, uint32_t sourceClockHz )`

Also initialize Smart card PHY interface .

This function ungates the Smart card clock, initializes the module to EMV default settings, configures the IRQ state structure, and enables the module-level interrupt to the core. Initialize RTOS synchronization objects and context.

Parameters

<code>base</code>	The Smart card peripheral base address.
<code>ctx</code>	The Smart card RTOS structure.
<code>sourceClockHz</code>	Smart card clock generation module source clock.

Returns

An zero in Success or error code.

#### 32.9.4.2 `int SMARTCARD_RTOS_Deinit ( rtos_smartcard_context_t * ctx )`

Deactivates also Smart card PHY interface, stops Smart card clocks. Free all synchronization objects allocated in RTOS Smart card context.

## Smart Card µCOS/II Driver

Parameters

<i>ctx</i>	The Smart card RTOS state.
------------	----------------------------

Returns

An zero in Success or error code.

### 32.9.4.3 int SMARTCARD\_RTOS\_Transfer ( *rtos\_smartcard\_context\_t \* ctx*, *smartcard\_xfer\_t \* xfer* )

A blocking (also known as synchronous) function means that the function returns after the transfer is done. User can cancel this transfer by calling function AbortTransfer.

Parameters

<i>ctx</i>	A pointer to the RTOS Smart card driver context.
<i>xfer</i>	Smart card transfer structure.

Returns

An zero in Success or error code.

### 32.9.4.4 int SMARTCARD\_RTOS\_WaitForXevent ( *rtos\_smartcard\_context\_t \* ctx* )

Task waits on the transfer finish event. Don't initialize transfer, just wait for transfer callback. Can be used while waiting on initial TS character.

Parameters

<i>ctx</i>	A pointer to the RTOS Smart card driver context.
------------	--

Returns

An zero in Success or error code.

### 32.9.4.5 int SMARTCARD\_RTOS\_Control ( *rtos\_smartcard\_context\_t \* ctx*, *smartcard\_control\_t control*, *uint32\_t param* )

Parameters

<i>ctx</i>	The Smart card RTOS context pointer.
<i>control</i>	Control type
<i>param</i>	Integer value of specific to control command.

Returns

An zero in Success or error code.

#### 32.9.4.6 int SMARTCARD\_RTOS\_PHY\_Control ( *rtos\_smartcard\_context\_t \* ctx*, *smartcard\_interface\_control\_t control*, *uint32\_t param* )

Parameters

<i>ctx</i>	The Smart card RTOS context pointer.
<i>control</i>	Control type
<i>param</i>	Integer value of specific to control command.

Returns

An zero in Success or error code.

#### 32.9.4.7 int SMARTCARD\_RTOS\_PHY\_Activate ( *rtos\_smartcard\_context\_t \* ctx*, *smartcard\_reset\_type\_t resetType* )

Parameters

<i>ctx</i>	The Smart card RTOS driver context structure.
<i>resetType</i>	type of reset to be performed, possible values = kSmartcardColdReset, kSmartcard-WarmReset

Returns

An zero in Success or error code.

#### 32.9.4.8 int SMARTCARD\_RTOS\_PHY\_Deactivate ( *rtos\_smartcard\_context\_t \* ctx* )

## Smart Card µCOS/II Driver

### Parameters

<i>ctx</i>	The Smart card RTOS driver context structure.
------------	---

### Returns

An zero in Success or error code.

## 32.10 Smart Card µCOS/III Driver

### 32.10.1 Overview

#### Data Structures

- struct `rtos_smartcard_context_t`  
*Runtime RTOS smart card driver context.* [More...](#)

#### Macros

- #define `RTOS_SMARTCARD_COMPLETE` 0x1u  
*Smart card RTOS transfer complete flag.*
- #define `RTOS_SMARTCARD_TIMEOUT` 0x2u  
*Smart card RTOS transfer time-out flag.*
- #define `SMARTCARD_Control`(base, context, control, param) `SMARTCARD_UART_Control`(base, context, control, 0)  
*Common Smart card driver API defines.*
- #define `SMARTCARD_Transfer`(base, context, xfer) `SMARTCARD_UART_TransferNonBlocking`(base, context, xfer)  
*Common Smart card API macro.*
- #define `SMARTCARD_Init`(base, context, sourceClockHz) `SMARTCARD_UART_Init`(base, context, sourceClockHz)  
*Common Smart card API macro.*
- #define `SMARTCARD_Deinit`(base) `SMARTCARD_UART_Deinit`(base)  
*Common Smart card API macro.*
- #define `SMARTCARD_GetTransferRemainingBytes`(base, context) `SMARTCARD_UART_GetTransferRemainingBytes`(base, context)  
*Common Smart card API macro.*
- #define `SMARTCARD_GetDefaultConfig`(cardParams) `SMARTCARD_UART_GetDefaultConfig`(cardParams)  
*Common Smart card API macro.*
- #define `SMARTCARD_PHY_Activate`(base, context, resetType) `SMARTCARD_PHY_GPIO_Activate`(base, context, resetType)  
*Common Smart card API macro.*
- #define `SMARTCARD_PHY_Deactivate`(base, context) `SMARTCARD_PHY_GPIO_Deactivate`(base, context)  
*Common Smart card API macro.*
- #define `SMARTCARD_PHY_Control`(base, context, control, param) `SMARTCARD_PHY_GPIO_Control`(base, context, control, param)  
*Common Smart card API macro.*
- #define `SMARTCARD_PHY_Init`(base, config, sourceClockHz) `SMARTCARD_PHY_GPIO_Init`(base, config, sourceClockHz)  
*Common Smart card API macro.*
- #define `SMARTCARD_PHY_Deinit`(base, config) `SMARTCARD_PHY_GPIO_Deinit`(base, config)  
*Common Smart card API macro.*

## Smart Card µCOS/III Driver

- #define **SMARTCARD\_PHY\_GetDefaultConfig**(config) **SMARTCARD\_PHY\_GPIO\_GetDefaultConfig**(config)  
*Common Smart card API macro.*

## Functions

- int **SMARTCARD\_RTOS\_Init** (void \*base, rtos\_smartcard\_context\_t \*ctx, uint32\_t sourceClock-Hz)  
*Initializes an Smart card (EMVSIM/UART) peripheral for Smart card/ISO-7816 operation.*
- int **SMARTCARD\_RTOS\_Deinit** (rtos\_smartcard\_context\_t \*ctx)  
*This function disables the Smart card (EMVSIM/UART) interrupts, disables the transmitter and receiver, and flushes the FIFOs (for modules that support FIFOs) and gates Smart card clock in SIM.*
- int **SMARTCARD\_RTOS\_Transfer** (rtos\_smartcard\_context\_t \*ctx, smartcard\_xfer\_t \*xfer)  
*Transfers data using interrupts.*
- int **SMARTCARD\_RTOS\_WaitForXevent** (rtos\_smartcard\_context\_t \*ctx)  
*Waits until transfer is finished.*
- int **SMARTCARD\_RTOS\_Control** (rtos\_smartcard\_context\_t \*ctx, smartcard\_control\_t control, uint32\_t param)  
*Controls Smart card module as per different user request.*
- int **SMARTCARD\_RTOS\_PHY\_Control** (rtos\_smartcard\_context\_t \*ctx, smartcard\_interface-control\_t control, uint32\_t param)  
*Controls Smart card module as per different user request.*
- int **SMARTCARD\_RTOS\_PHY\_Activate** (rtos\_smartcard\_context\_t \*ctx, smartcard\_reset\_type\_t resetType)  
*Activates the Smart card interface.*
- int **SMARTCARD\_RTOS\_PHY\_Deactivate** (rtos\_smartcard\_context\_t \*ctx)  
*Deactivates the Smart card interface.*

### 32.10.2 Data Structure Documentation

#### 32.10.2.1 struct rtos\_smartcard\_context\_t

Runtime RTOS Smart card driver context.

#### Data Fields

- SemaphoreHandle\_t **x\_sem**  
*RTOS unique access assurance object.*
- EventGroupHandle\_t **x\_event**  
*RTOS synchronization object.*
- smartcard\_context\_t **x\_context**  
*transactional layer state*
- OS\_EVENT \* **x\_sem**  
*RTOS unique access assurance object.*
- OS\_FLAG\_GRP \* **x\_event**  
*RTOS synchronization object.*

- OS\_SEM `x_sem`  
*RTOS unique access assurance object.*
- OS\_FLAG\_GRP `x_event`  
*RTOS synchronization object.*

### 32.10.2.1.0.18 Field Documentation

#### 32.10.2.1.0.18.1 `smartcard_context_t rtos_smartcard_context_t::x_context`

Transactional layer state.

### 32.10.3 Macro Definition Documentation

#### 32.10.3.1 `#define SMARTCARD_Control( base, context, control, param ) SMARTCARD_UART_Control(base, context, control, 0)`

Common Smart card API macro

### 32.10.4 Function Documentation

#### 32.10.4.1 `int SMARTCARD_RTOS_Init ( void * base, rtos_smartcard_context_t * ctx, uint32_t sourceClockHz )`

Also initialize Smart card PHY interface .

This function ungates the Smart card clock, initializes the module to EMV default settings, configures the IRQ state structure, and enables the module-level interrupt to the core. Initialize RTOS synchronization objects and context.

Parameters

<code>base</code>	The Smart card peripheral base address.
<code>ctx</code>	The Smart card RTOS structure.
<code>sourceClockHz</code>	Smart card clock generation module source clock.

Returns

An zero in Success or error code.

#### 32.10.4.2 `int SMARTCARD_RTOS_Deinit ( rtos_smartcard_context_t * ctx )`

Deactivates also Smart card PHY interface, stops Smart card clocks. Free all synchronization objects allocated in RTOS Smart card context.

## Smart Card µCOS/III Driver

Parameters

<i>ctx</i>	The Smart card RTOS state.
------------	----------------------------

Returns

An zero in Success or error code.

### 32.10.4.3 int SMARTCARD\_RRTOS\_Transfer ( *rtos\_smartcard\_context\_t \* ctx*, *smartcard\_xfer\_t \* xfer* )

A blocking (also known as synchronous) function means that the function returns after the transfer is done. User can cancel this transfer by calling function AbortTransfer.

Parameters

<i>ctx</i>	A pointer to the RTOS Smart card driver context.
<i>xfer</i>	Smart card transfer structure.

Returns

An zero in Success or error code.

### 32.10.4.4 int SMARTCARD\_RRTOS\_WaitForXevent ( *rtos\_smartcard\_context\_t \* ctx* )

Task waits on the transfer finish event. Don't initialize transfer, just wait for transfer callback. Can be used while waiting on initial TS character.

Parameters

<i>ctx</i>	A pointer to the RTOS Smart card driver context.
------------	--

Returns

An zero in Success or error code.

### 32.10.4.5 int SMARTCARD\_RRTOS\_Control ( *rtos\_smartcard\_context\_t \* ctx*, *smartcard\_control\_t control*, *uint32\_t param* )

Parameters

<i>ctx</i>	The Smart card RTOS context pointer.
<i>control</i>	Control type
<i>param</i>	Integer value of specific to control command.

Returns

An zero in Success or error code.

#### 32.10.4.6 int SMARTCARD\_RTOS\_PHY\_Control ( *rtos\_smartcard\_context\_t \* ctx*, *smartcard\_interface\_control\_t control*, *uint32\_t param* )

Parameters

<i>ctx</i>	The Smart card RTOS context pointer.
<i>control</i>	Control type
<i>param</i>	Integer value of specific to control command.

Returns

An zero in Success or error code.

#### 32.10.4.7 int SMARTCARD\_RTOS\_PHY\_Activate ( *rtos\_smartcard\_context\_t \* ctx*, *smartcard\_reset\_type\_t resetType* )

Parameters

<i>ctx</i>	The Smart card RTOS driver context structure.
<i>resetType</i>	type of reset to be performed, possible values = kSmartcardColdReset, kSmartcardWarmReset

Returns

An zero in Success or error code.

#### 32.10.4.8 int SMARTCARD\_RTOS\_PHY\_Deactivate ( *rtos\_smartcard\_context\_t \* ctx* )

## Smart Card µCOS/III Driver

### Parameters

<i>ctx</i>	The Smart card RTOS driver context structure.
------------	---

### Returns

An zero in Success or error code.

# Chapter 33

## SMC: System Mode Controller Driver

### 33.1 Overview

The KSDK provides a Peripheral driver for the System Mode Controller (SMC) module of Kinetis devices. The SMC module is responsible for sequencing the system into and out of all low-power Stop and Run modes.

API functions are provided for configuring the system working in a dedicated power mode. For different power modes, function `SMC_SetPowerModeXXX` accepts different parameters. System power mode state transitions are not available for between power modes. For details about available transitions, see the Power mode transitions section in the SoC reference manual.

### Enumerations

- enum `smc_power_mode_protection_t` {  
    `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` }  
*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` }  
*Stop mode definition.*
- enum `smc_partial_stop_option_t` {  
    `kSMC_PartialStop` = `0U`,  
    `kSMC_PartialStop1` = `1U`,  
    `kSMC_PartialStop2` = `2U` }  
*Partial STOP option.*
- enum `_smc_status` { `kStatus_SMC_StopAbort` = `MAKE_STATUS(kStatusGroup_POWER, 0)` }  
*SMC configuration status.*

## Enumeration Type Documentation

### Driver version

- #define **FSL\_SMC\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 2))  
*SMC driver version 2.0.2.*

### 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.*
- status\_t **SMC\_SetPowerModeRun** (SMC\_Type \*base)  
*Configure the system to RUN power mode.*
- status\_t **SMC\_SetPowerModeWait** (SMC\_Type \*base)  
*Configure the system to WAIT power mode.*
- status\_t **SMC\_SetPowerModeStop** (SMC\_Type \*base, **smc\_partial\_stop\_option\_t** option)  
*Configure the system to Stop power mode.*
- status\_t **SMC\_SetPowerModeVlpr** (SMC\_Type \*base)  
*Configure the system to VLPR power mode.*
- status\_t **SMC\_SetPowerModeVlpw** (SMC\_Type \*base)  
*Configure the system to VLPW power mode.*
- status\_t **SMC\_SetPowerModeVlps** (SMC\_Type \*base)  
*Configure the system to VLPS power mode.*

### 33.2 Macro Definition Documentation

#### 33.2.1 #define FSL\_SMC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))

### 33.3 Enumeration Type Documentation

#### 33.3.1 enum smc\_power\_mode\_protection\_t

Enumerator

**kSMC\_AllowPowerModeVlp** Allow Very-Low-Power Mode.

**kSMC\_AllowPowerModeAll** Allow all power mode.

#### 33.3.2 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

### 33.3.3 enum smc\_run\_mode\_t

Enumerator

*kSMC\_RunNormal* normal RUN mode.

*kSMC\_RunVlpr* Very-Low-Power RUN mode.

### 33.3.4 enum smc\_stop\_mode\_t

Enumerator

*kSMC\_StopNormal* Normal STOP mode.

*kSMC\_StopVlps* Very-Low-Power STOP mode.

### 33.3.5 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.

### 33.3.6 enum \_smc\_status

Enumerator

*kStatus\_SMC\_StopAbort* Entering Stop mode is abort.

## 33.4 Function Documentation

### 33.4.1 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)*.

## Function Documentation

Parameters

<i>base</i>	SMC peripheral base address.
<i>allowedModes</i>	Bitmap of the allowed power modes.

### 33.4.2 static smc\_power\_state\_t SMC\_GetPowerModeState ( SMC\_Type \* *base* ) [inline], [static]

This function returns the current power mode stat. Once application switches the power mode, it should always check the stat to check whether it runs into the specified mode or not. An 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 stat.

Parameters

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

Returns

Current power mode status.

### 33.4.3 status\_t SMC\_SetPowerModeRun ( SMC\_Type \* *base* )

Parameters

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

Returns

SMC configuration error code.

### 33.4.4 status\_t SMC\_SetPowerModeWait ( SMC\_Type \* *base* )

Parameters

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

Returns

SMC configuration error code.

### 33.4.5 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.

### 33.4.6 status\_t SMC\_SetPowerModeVlpr ( SMC\_Type \* *base* )

Parameters

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

Returns

SMC configuration error code.

### 33.4.7 status\_t SMC\_SetPowerModeVlpw ( SMC\_Type \* *base* )

Parameters

## Function Documentation

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

Returns

SMC configuration error code.

### 33.4.8 **status\_t SMC\_SetPowerModeVlps ( SMC\_Type \* *base* )**

Parameters

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

Returns

SMC configuration error code.

## Chapter 34

# TPM: Timer PWM Module

### 34.1 Overview

The KSDK provides a driver for the Timer PWM Module (TPM) of Kinetis devices.

The KSDK TPM driver supports the generation of PWM signals, input capture, and output compare modes. On some SoC's, the driver supports the generation of combined PWM signals, dual-edge capture, and quadrature decode modes. The driver also supports configuring each of the TPM fault inputs. The fault input is available only on some SoC's.

The function [TPM\\_Init\(\)](#) initializes the TPM with specified configurations. The function [TPM\\_GetDefaultConfig\(\)](#) gets the default configurations. On some SoC's, the initialization function issues a software reset to reset the TPM internal logic. The initialization function configures the TPM's behavior when it receives a trigger input and its operation in doze and debug modes.

The function [TPM\\_Deinit\(\)](#) disables the TPM counter and turns off the module clock.

The function [TPM\\_SetupPwm\(\)](#) sets up TPM channels for the PWM output. The function can set up the PWM signal properties for multiple channels. Each channel has its own [tpm\\_chnl\\_pwm\\_signal\\_param\\_t](#) structure that is used to specify the output signals duty cycle and level-mode. However, the same PWM period and PWM mode is applied to all channels requesting a PWM output. The signal duty cycle is provided as a percentage of the PWM period. Its value should be between 0 and 100 where 0=inactive signal (0% duty cycle) and 100=always active signal (100% duty cycle). When generating a combined PWM signal, the channel number passed refers to a channel pair number, for example 0 refers to channel 0 and 1, 1 refers to channels 2 and 3.

The function [TPM\\_UpdatePwmDutycycle\(\)](#) updates the PWM signal duty cycle of a particular TPM channel.

The function [TPM\\_UpdateChnlEdgeLevelSelect\(\)](#) updates the level select bits of a particular TPM channel. This can be used to disable the PWM output when making changes to the PWM signal.

The function [TPM\\_SetupInputCapture\(\)](#) sets up a TPM channel for input capture. The user can specify the capture edge.

The function [TPM\\_SetupDualEdgeCapture\(\)](#) can be used to measure the pulse width of a signal. This is available only for certain SoC's. A channel pair is used during the capture with the input signal coming through a channel that can be configured. The user can specify the capture edge for each channel and any filter value to be used when processing the input signal.

The function [TPM\\_SetupOutputCompare\(\)](#) sets up a TPM channel for output comparison. The user can specify the channel output on a successful comparison and a comparison value.

The function [TPM\\_SetupQuadDecode\(\)](#) sets up TPM channels 0 and 1 for quad decode, which is available only for certain SoC's. The user can specify the quad decode mode, polarity, and filter properties for each input signal.

## Typical use case

The function TPM\_SetupFault() sets up the properties for each fault, which is available only for certain SoC's. The user can specify the fault polarity and whether to use a filter on a fault input. The overall fault filter value and fault control mode are set up during initialization.

Provides functions to get and clear the TPM status.

Provides functions to enable/disable TPM interrupts and get current enabled interrupts.

## 34.2 Typical use case

### 34.2.1 PWM output

Output the PWM signal on 2 TPM channels with different duty cycles. Periodically update the PWM signal duty cycle.

```
int main(void)
{
    bool brightnessUp = true; /* Indicates whether the LED is brighter or dimmer. */
    tpm_config_t tpmInfo;
    uint8_t updatedDutycycle = 0U;
    tpm_chnl_pwm_signal_param_t tpmParam[2];

    /* Configures the TPM parameters with frequency 24 kHz. */
    tpmParam[0].chnlNumber = (tpm_chnl_t)BOARD_FIRST_TPM_CHANNEL;
    tpmParam[0].level = kTPM_LowTrue;
    tpmParam[0].dutyCyclePercent = 0U;

    tpmParam[1].chnlNumber = (tpm_chnl_t)BOARD_SECOND_TPM_CHANNEL;
    tpmParam[1].level = kTPM_LowTrue;
    tpmParam[1].dutyCyclePercent = 0U;

    /* Board pin, clock, and debug console initialization. */
    BOARD_InitHardware();

    TPM_GetDefaultConfig(&tpmInfo);
    /* Initializes the TPM module. */
    TPM_Init(BOARD_TPM_BASEADDR, &tpmInfo);

    TPM_SetupPwm(BOARD_TPM_BASEADDR, tpmParam, 2U,
                 kTPM_EdgeAlignedPwm, 24000U, TPM_SOURCE_CLOCK);
    TPM_StartTimer(BOARD_TPM_BASEADDR, kTPM_SystemClock);
    while (1)
    {
        /* Delays to see the change of LED brightness. */
        delay();

        if (brightnessUp)
        {
            /* Increases a duty cycle until it reaches a limited value. */
            if (++updatedDutycycle == 100U)
            {
                brightnessUp = false;
            }
        }
        else
        {
            /* Decreases a duty cycle until it reaches a limited value. */
            if (--updatedDutycycle == 0U)
            {
                brightnessUp = true;
            }
        }
    }
}
```

```

    /* Starts PWM mode with an updated duty cycle. */
    TPM_UpdatePwmDutycycle(BOARD_TPM_BASEADDR, (
        tpm_chnl_t)BOARD_FIRST_TPM_CHANNEL, kTPM_EdgeAlignedPwm,
        updatedDutycycle);
    TPM_UpdatePwmDutycycle(BOARD_TPM_BASEADDR, (
        tpm_chnl_t)BOARD_SECOND_TPM_CHANNEL, kTPM_EdgeAlignedPwm,
        updatedDutycycle);
}
}

```

## Data Structures

- struct [tpm\\_chnl\\_pwm\\_signal\\_param\\_t](#)  
*Options to configure a TPM channel's PWM signal.* [More...](#)
- struct [tpm\\_config\\_t](#)  
*TPM config structure.* [More...](#)

## Enumerations

- enum [tpm\\_chnl\\_t](#) {
 kTPM\_Chnl\_0 = 0U,
 kTPM\_Chnl\_1,
 kTPM\_Chnl\_2,
 kTPM\_Chnl\_3,
 kTPM\_Chnl\_4,
 kTPM\_Chnl\_5,
 kTPM\_Chnl\_6,
 kTPM\_Chnl\_7 }
   
*List of TPM channels.*
- enum [tpm\\_pwm\\_mode\\_t](#) {
 kTPM\_EdgeAlignedPwm = 0U,
 kTPM\_CenterAlignedPwm }
   
*TPM PWM operation modes.*
- enum [tpm\\_pwm\\_level\\_select\\_t](#) {
 kTPM\_NoPwmSignal = 0U,
 kTPM\_LowTrue,
 kTPM\_HighTrue }
   
*TPM PWM output pulse mode: high-true, low-true or no output.*
- enum [tpm\\_trigger\\_select\\_t](#)
  
*Trigger options available.*
- enum [tpm\\_output\\_compare\\_mode\\_t](#) {
 kTPM\_NoOutputSignal = (1U << TPM\_CnSC\_MSA\_SHIFT),
 kTPM\_ToggleOnMatch = ((1U << TPM\_CnSC\_MSA\_SHIFT) | (1U << TPM\_CnSC\_ELSA\_S-HIFT)),
 kTPM\_ClearOnMatch = ((1U << TPM\_CnSC\_MSA\_SHIFT) | (2U << TPM\_CnSC\_ELSA\_S-HIFT)),
 kTPM\_SetOnMatch = ((1U << TPM\_CnSC\_MSA\_SHIFT) | (3U << TPM\_CnSC\_ELSA\_SHIF-T)),
 kTPM\_HighPulseOutput = ((3U << TPM\_CnSC\_MSA\_SHIFT) | (1U << TPM\_CnSC\_ELSA\_-)}

## Typical use case

- ```
SHIFT)),  
kTPM_LowPulseOutput = ((3U << TPM_CnSC_MSA_SHIFT) | (2U << TPM_CnSC_ELSA_S-  
HIFT)) }  
    TPM output compare modes.  
• enum tpm_input_capture_edge_t {  
    kTPM_RisingEdge = (1U << TPM_CnSC_ELSA_SHIFT),  
    kTPM_FallingEdge = (2U << TPM_CnSC_ELSA_SHIFT),  
    kTPM_RiseAndFallEdge = (3U << TPM_CnSC_ELSA_SHIFT) }  
    TPM input capture edge.  
• enum tpm_clock_source_t {  
    kTPM_SystemClock = 1U,  
    kTPM_ExternalClock }  
    TPM clock source selection.  
• enum tpm_clock_prescale_t {  
    kTPM_Prescale_Divide_1 = 0U,  
    kTPM_Prescale_Divide_2,  
    kTPM_Prescale_Divide_4,  
    kTPM_Prescale_Divide_8,  
    kTPM_Prescale_Divide_16,  
    kTPM_Prescale_Divide_32,  
    kTPM_Prescale_Divide_64,  
    kTPM_Prescale_Divide_128 }  
    TPM prescale value selection for the clock source.  
• enum tpm_interrupt_enable_t {  
    kTPM_Chnl0InterruptEnable = (1U << 0),  
    kTPM_Chnl1InterruptEnable = (1U << 1),  
    kTPM_Chnl2InterruptEnable = (1U << 2),  
    kTPM_Chnl3InterruptEnable = (1U << 3),  
    kTPM_Chnl4InterruptEnable = (1U << 4),  
    kTPM_Chnl5InterruptEnable = (1U << 5),  
    kTPM_Chnl6InterruptEnable = (1U << 6),  
    kTPM_Chnl7InterruptEnable = (1U << 7),  
    kTPM_TimeOverflowInterruptEnable = (1U << 8) }  
    List of TPM interrupts.  
• enum tpm_status_flags_t {  
    kTPM_Chnl0Flag = (1U << 0),  
    kTPM_Chnl1Flag = (1U << 1),  
    kTPM_Chnl2Flag = (1U << 2),  
    kTPM_Chnl3Flag = (1U << 3),  
    kTPM_Chnl4Flag = (1U << 4),  
    kTPM_Chnl5Flag = (1U << 5),  
    kTPM_Chnl6Flag = (1U << 6),  
    kTPM_Chnl7Flag = (1U << 7),  
    kTPM_TimeOverflowFlag = (1U << 8) }  
    List of TPM flags.
```

## Driver version

- #define **FSL TPM\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 2))  
*Version 2.0.2.*

## Initialization and deinitialization

- void **TPM\_Init** (TPM\_Type \*base, const **tpm\_config\_t** \*config)  
*Ungates the TPM clock and configures the peripheral for basic operation.*
- void **TPM\_Deinit** (TPM\_Type \*base)  
*Stops the counter and gates the TPM clock.*
- void **TPM\_GetDefaultConfig** (**tpm\_config\_t** \*config)  
*Fill in the TPM config struct with the default settings.*

## Channel mode operations

- status\_t **TPM\_SetupPwm** (TPM\_Type \*base, const **tpm\_chnl\_pwm\_signal\_param\_t** \*chnlParams, uint8\_t numOfChnls, **tpm\_pwm\_mode\_t** mode, uint32\_t pwmFreq\_Hz, uint32\_t srcClock\_Hz)  
*Configures the PWM signal parameters.*
- void **TPM\_UpdatePwmDutyCycle** (TPM\_Type \*base, **tpm\_chnl\_t** chnlNumber, **tpm\_pwm\_mode\_t** currentPwmMode, uint8\_t dutyCyclePercent)  
*Update the duty cycle of an active PWM signal.*
- void **TPM\_UpdateChnlEdgeLevelSelect** (TPM\_Type \*base, **tpm\_chnl\_t** chnlNumber, uint8\_t level)  
*Update the edge level selection for a channel.*
- void **TPM\_SetupInputCapture** (TPM\_Type \*base, **tpm\_chnl\_t** chnlNumber, **tpm\_input\_capture\_edge\_t** captureMode)  
*Enables capturing an input signal on the channel using the function parameters.*
- void **TPM\_SetupOutputCompare** (TPM\_Type \*base, **tpm\_chnl\_t** chnlNumber, **tpm\_output\_compare\_mode\_t** compareMode, uint32\_t compareValue)  
*Configures the TPM to generate timed pulses.*

## Interrupt Interface

- void **TPM\_EnableInterrupts** (TPM\_Type \*base, uint32\_t mask)  
*Enables the selected TPM interrupts.*
- void **TPM\_DisableInterrupts** (TPM\_Type \*base, uint32\_t mask)  
*Disables the selected TPM interrupts.*
- uint32\_t **TPM\_GetEnabledInterrupts** (TPM\_Type \*base)  
*Gets the enabled TPM interrupts.*

## Status Interface

- static uint32\_t **TPM\_GetStatusFlags** (TPM\_Type \*base)  
*Gets the TPM status flags.*
- static void **TPM\_ClearStatusFlags** (TPM\_Type \*base, uint32\_t mask)  
*Clears the TPM status flags.*

## Timer Start and Stop

- static void **TPM\_StartTimer** (TPM\_Type \*base, **tpm\_clock\_source\_t** clockSource)

## Data Structure Documentation

- static void [TPM\\_StopTimer](#) (TPM\_Type \*base)  
*Starts the TPM counter.*
- static void [TPM\\_StopTimer](#) (TPM\_Type \*base)  
*Stops the TPM counter.*

### 34.3 Data Structure Documentation

#### 34.3.1 struct tpm\_chnl\_pwm\_signal\_param\_t

##### Data Fields

- [tpm\\_chnl\\_t chnlNumber](#)  
*TPM channel to configure.*
- [tpm\\_pwm\\_level\\_select\\_t level](#)  
*PWM output active level select.*
- [uint8\\_t dutyCyclePercent](#)  
*PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)...*

##### 34.3.1.0.0.19 Field Documentation

###### 34.3.1.0.0.19.1 [tpm\\_chnl\\_t tpm\\_chnl\\_pwm\\_signal\\_param\\_t::chnlNumber](#)

In combined mode (available in some SoC's, this represents the channel pair number

###### 34.3.1.0.0.19.2 [uint8\\_t tpm\\_chnl\\_pwm\\_signal\\_param\\_t::dutyCyclePercent](#)

100=always active signal (100% duty cycle)

#### 34.3.2 struct tpm\_config\_t

This structure holds the configuration settings for the TPM peripheral. To initialize this structure to reasonable defaults, call the [TPM\\_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

- [tpm\\_clock\\_prescale\\_t prescale](#)  
*Select TPM clock prescale value.*
- [bool useGlobalTimeBase](#)  
*true: Use of an external global time base is enabled; false: disabled*
- [tpm\\_trigger\\_select\\_t triggerSelect](#)  
*Input trigger to use for controlling the counter operation.*
- [bool enableDoze](#)  
*true: TPM counter is paused in doze mode; false: TPM counter continues in doze mode*
- [bool enableDebugMode](#)  
*true: TPM counter continues in debug mode; false: TPM counter is paused in debug mode*

- bool `enableReloadOnTrigger`  
*true: TPM counter is reloaded on trigger; false: TPM counter not reloaded*
- bool `enableStopOnOverflow`  
*true: TPM counter stops after overflow; false: TPM counter continues running after overflow*
- bool `enableStartOnTrigger`  
*true: TPM counter only starts when a trigger is detected; false: TPM counter starts immediately*

## 34.4 Enumeration Type Documentation

### 34.4.1 enum tpm\_chnl\_t

Note

Actual number of available channels is SoC dependent

Enumerator

- `kTPM_Chnl_0`** TPM channel number 0.
- `kTPM_Chnl_1`** TPM channel number 1.
- `kTPM_Chnl_2`** TPM channel number 2.
- `kTPM_Chnl_3`** TPM channel number 3.
- `kTPM_Chnl_4`** TPM channel number 4.
- `kTPM_Chnl_5`** TPM channel number 5.
- `kTPM_Chnl_6`** TPM channel number 6.
- `kTPM_Chnl_7`** TPM channel number 7.

### 34.4.2 enum tpm\_pwm\_mode\_t

Enumerator

- `kTPM_EdgeAlignedPwm`** Edge aligned PWM.
- `kTPM_CenterAlignedPwm`** Center aligned PWM.

### 34.4.3 enum tpm\_pwm\_level\_select\_t

Enumerator

- `kTPM_NoPwmSignal`** No PWM output on pin.
- `kTPM_LowTrue`** Low true pulses.
- `kTPM_HighTrue`** High true pulses.

## Enumeration Type Documentation

### 34.4.4 enum tpm\_trigger\_select\_t

This is used for both internal & external trigger sources (external option available in certain SoC's)

Note

The actual trigger options available is SoC-specific.

### 34.4.5 enum tpm\_output\_compare\_mode\_t

Enumerator

*kTPM\_NoOutputSignal* No channel output when counter reaches CnV.

*kTPM\_ToggleOnMatch* Toggle output.

*kTPM\_ClearOnMatch* Clear output.

*kTPM\_SetOnMatch* Set output.

*kTPM\_HighPulseOutput* Pulse output high.

*kTPM\_LowPulseOutput* Pulse output low.

### 34.4.6 enum tpm\_input\_capture\_edge\_t

Enumerator

*kTPM\_RisingEdge* Capture on rising edge only.

*kTPM\_FallingEdge* Capture on falling edge only.

*kTPM\_RiseAndFallEdge* Capture on rising or falling edge.

### 34.4.7 enum tpm\_clock\_source\_t

Enumerator

*kTPM\_SystemClock* System clock.

*kTPM\_ExternalClock* External clock.

### 34.4.8 enum tpm\_clock\_prescale\_t

Enumerator

*kTPM\_Prescale\_Divide\_1* Divide by 1.

*kTPM\_Prescale\_Divide\_2* Divide by 2.

*kTPM\_Prescale\_Divide\_4* Divide by 4.  
*kTPM\_Prescale\_Divide\_8* Divide by 8.  
*kTPM\_Prescale\_Divide\_16* Divide by 16.  
*kTPM\_Prescale\_Divide\_32* Divide by 32.  
*kTPM\_Prescale\_Divide\_64* Divide by 64.  
*kTPM\_Prescale\_Divide\_128* Divide by 128.

### 34.4.9 enum tpm\_interrupt\_enable\_t

Enumerator

*kTPM\_Chnl0InterruptEnable* Channel 0 interrupt.  
*kTPM\_Chnl1InterruptEnable* Channel 1 interrupt.  
*kTPM\_Chnl2InterruptEnable* Channel 2 interrupt.  
*kTPM\_Chnl3InterruptEnable* Channel 3 interrupt.  
*kTPM\_Chnl4InterruptEnable* Channel 4 interrupt.  
*kTPM\_Chnl5InterruptEnable* Channel 5 interrupt.  
*kTPM\_Chnl6InterruptEnable* Channel 6 interrupt.  
*kTPM\_Chnl7InterruptEnable* Channel 7 interrupt.  
*kTPM\_TimeOverflowInterruptEnable* Time overflow interrupt.

### 34.4.10 enum tpm\_status\_flags\_t

Enumerator

*kTPM\_Chnl0Flag* Channel 0 flag.  
*kTPM\_Chnl1Flag* Channel 1 flag.  
*kTPM\_Chnl2Flag* Channel 2 flag.  
*kTPM\_Chnl3Flag* Channel 3 flag.  
*kTPM\_Chnl4Flag* Channel 4 flag.  
*kTPM\_Chnl5Flag* Channel 5 flag.  
*kTPM\_Chnl6Flag* Channel 6 flag.  
*kTPM\_Chnl7Flag* Channel 7 flag.  
*kTPM\_TimeOverflowFlag* Time overflow flag.

## 34.5 Function Documentation

### 34.5.1 void TPM\_Init ( TPM\_Type \* *base*, const tpm\_config\_t \* *config* )

Note

This API should be called at the beginning of the application using the TPM driver.

## Function Documentation

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | TPM peripheral base address             |
| <i>config</i> | Pointer to user's TPM config structure. |

### 34.5.2 void TPM\_Deinit ( TPM\_Type \* *base* )

Parameters

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

### 34.5.3 void TPM\_GetDefaultConfig ( tpm\_config\_t \* *config* )

The default values are:

```
*     config->prescale = kTPM_Prescale_Divide_1;
*     config->useGlobalTimeBase = false;
*     config->dozeEnable = false;
*     config->dbgMode = false;
*     config->enableReloadOnTrigger = false;
*     config->enableStopOnOverflow = false;
*     config->enableStartOnTrigger = false;
*#if FSL_FEATURE TPM HAS_PAUSE_COUNTER_ON_TRIGGER
*     config->enablePauseOnTrigger = false;
#endif
*     config->triggerSelect = kTPM_Trigger_Select_0;
*#if FSL_FEATURE TPM HAS_EXTERNAL_TRIGGER_SELECTION
*     config->triggerSource = kTPM_TriggerSource_External;
#endif
*
```

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>config</i> | Pointer to user's TPM config structure. |
|---------------|-----------------------------------------|

### 34.5.4 status\_t TPM\_SetupPwm ( TPM\_Type \* *base*, const tpm\_chnl\_pwm\_signal\_param\_t \* *chnlParams*, uint8\_t *numOfChnls*, tpm\_pwm\_mode\_t *mode*, uint32\_t *pwmFreq\_Hz*, uint32\_t *srcClock\_Hz* )

User calls this function to configure the PWM signals period, mode, dutycycle and edge. Use this function to configure all the TPM channels that will be used to output a PWM signal

Parameters

|                    |                                                                                     |
|--------------------|-------------------------------------------------------------------------------------|
| <i>base</i>        | TPM peripheral base address                                                         |
| <i>chnlParams</i>  | Array of PWM channel parameters to configure the channel(s)                         |
| <i>numOfChnls</i>  | Number of channels to configure, this should be the size of the array passed in     |
| <i>mode</i>        | PWM operation mode, options available in enumeration <a href="#">tpm_pwm_mode_t</a> |
| <i>pwmFreq_Hz</i>  | PWM signal frequency in Hz                                                          |
| <i>srcClock_Hz</i> | TPM counter clock in Hz                                                             |

Returns

kStatus\_Success if the PWM setup was successful, kStatus\_Error on failure

**34.5.5 void TPM\_UpdatePwmDutycycle ( **TPM\_Type** \* *base*, **tpm\_chnl\_t** *chnlNumber*, **tpm\_pwm\_mode\_t** *currentPwmMode*, **uint8\_t** *dutyCyclePercent* )**

Parameters

|                         |                                                                                                                               |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>             | TPM peripheral base address                                                                                                   |
| <i>chnlNumber</i>       | The channel number. In combined mode, this represents the channel pair number                                                 |
| <i>currentPwmMode</i>   | The current PWM mode set during PWM setup                                                                                     |
| <i>dutyCyclePercent</i> | New PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle) |

**34.5.6 void TPM\_UpdateChnlEdgeLevelSelect ( **TPM\_Type** \* *base*, **tpm\_chnl\_t** *chnlNumber*, **uint8\_t** *level* )**

Parameters

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

## Function Documentation

|                   |                                                                                                                                                       |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>chnlNumber</i> | The channel number                                                                                                                                    |
| <i>level</i>      | The level to be set to the ELSnB:ELSnA field; valid values are 00, 01, 10, 11. See the appropriate SoC reference manual for details about this field. |

### 34.5.7 void TPM\_SetupInputCapture ( **TPM\_Type** \* *base*, **tpm\_chnl\_t** *chnlNumber*, **tpm\_input\_capture\_edge\_t** *captureMode* )

When the edge specified in the captureMode argument occurs on the channel, the TPM counter is captured into the CnV register. The user has to read the CnV register separately to get this value.

Parameters

|                    |                                 |
|--------------------|---------------------------------|
| <i>base</i>        | TPM peripheral base address     |
| <i>chnlNumber</i>  | The channel number              |
| <i>captureMode</i> | Specifies which edge to capture |

### 34.5.8 void TPM\_SetupOutputCompare ( **TPM\_Type** \* *base*, **tpm\_chnl\_t** *chnlNumber*, **tpm\_output\_compare\_mode\_t** *compareMode*, **uint32\_t** *compareValue* )

When the TPM counter matches the value of compareVal argument (this is written into CnV reg), the channel output is changed based on what is specified in the compareMode argument.

Parameters

|                     |                                                                        |
|---------------------|------------------------------------------------------------------------|
| <i>base</i>         | TPM peripheral base address                                            |
| <i>chnlNumber</i>   | The channel number                                                     |
| <i>compareMode</i>  | Action to take on the channel output when the compare condition is met |
| <i>compareValue</i> | Value to be programmed in the CnV register.                            |

### 34.5.9 void TPM\_EnableInterrupts ( **TPM\_Type** \* *base*, **uint32\_t** *mask* )

Parameters

|             |                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | TPM peripheral base address                                                                                         |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">tpm_interrupt_enable_t</a> |

**34.5.10 void TPM\_DisableInterrupts ( TPM\_Type \* *base*, uint32\_t *mask* )**

Parameters

|             |                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | TPM peripheral base address                                                                                          |
| <i>mask</i> | The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">tpm_interrupt_enable_t</a> |

**34.5.11 uint32\_t TPM\_GetEnabledInterrupts ( TPM\_Type \* *base* )**

Parameters

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

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [tpm\\_interrupt\\_enable\\_t](#)

**34.5.12 static uint32\_t TPM\_GetStatusFlags ( TPM\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

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

**34.5.13 static void TPM\_ClearStatusFlags ( TPM\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

## Function Documentation

Parameters

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

**34.5.14 static void TPM\_StartTimer ( TPM\_Type \* *base*, tpm\_clock\_source\_t *clockSource* ) [inline], [static]**

Parameters

|                    |                                                                           |
|--------------------|---------------------------------------------------------------------------|
| <i>base</i>        | TPM peripheral base address                                               |
| <i>clockSource</i> | TPM clock source; once clock source is set the counter will start running |

**34.5.15 static void TPM\_StopTimer ( TPM\_Type \* *base* ) [inline], [static]**

Parameters

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

# Chapter 35

## TRNG: True Random Number Generator

### 35.1 Overview

The Kinetis SDK provides a peripheral driver for the True Random Number Generator (TRNG) module of Kinetis devices.

The True Random Number Generator is hardware accelerator module that generates a 512-bit entropy as needed by an entropy consuming module or by other post processing functions. A typical entropy consumer is a pseudo random number generator (PRNG) which can be implemented to achieve both true randomness and cryptographic strength random numbers using the TRNG output as its entropy seed. The entropy generated by a TRNG is intended for direct use by functions that generate secret keys, per-message secrets, random challenges, and other similar quantities used in cryptographic algorithms.

### 35.2 TRNG Initialization

1. Define the TRNG user configuration structure. Use `TRNG_InitUserConfigDefault()` function to set it to default TRNG configuration values.
2. Initialize the TRNG module, call the `TRNG_Init()` function and pass the user configuration structure. This function automatically enables the TRNG module and its clock. After that, the TRNG is enabled and the entropy generation starts working.
3. To disable the TRNG module, call the `TRNG_Deinit()` function.

### 35.3 Get random data from TRNG

1. `TRNG_GetRandomData()` function gets random data from the TRNG module.

This example code shows how to initialize and get random data from the TRNG driver:

```
{  
    trng_user_config_t    trngConfig;  
    status_t              status;  
    uint32_t              data;  
  
    /* Initialize TRNG configuration structure to default */  
    TRNG_InitUserConfigDefault(&trngConfig);  
  
    /* Initialize TRNG */  
    status = TRNG_Init(TRNG0, &trngConfig);  
  
    if (status == kStatus_Success)  
    {  
        /* Read Random data */  
        if ((status = TRNG_GetRandomData(TRNG0, data, sizeof(data))) ==  
            kStatus_TRNG_Success)  
        {  
            /* Print data */  
            PRINTF("Random = 0x%X\r\n", i, data );  
  
            PRINTF ("Succeed.\r\n");  
        }  
    }  
}
```

## Get random data from TRNG

```
    }
    else
    {
        PRINTF ("TRNG failed! (0x%x)\r\n", status);
    }

    /* Deinitialize TRNG*/
    TRNG_Deinit(TRNG0);
}
else
{
    PRINTF("TRNG initialization failed!\r\n");
}
}
```

## Data Structures

- struct `trng_statistical_check_limit_t`  
*Data structure for definition of statistical check limits.* [More...](#)
- struct `trng_config_t`  
*Data structure for the TRNG initialization.* [More...](#)

## Enumerations

- enum `trng_sample_mode_t` {  
    kTRNG\_SampleModeVonNeumann = 0U,  
    kTRNG\_SampleModeRaw = 1U,  
    kTRNG\_SampleModeVonNeumannRaw }  
*TRNG sample mode.*
- enum `trng_clock_mode_t` {  
    kTRNG\_ClockModeRingOscillator = 0U,  
    kTRNG\_ClockModeSystem = 1U }  
*TRNG clock mode.*
- enum `trng_ring_osc_div_t` {  
    kTRNG\_RingOscDiv0 = 0U,  
    kTRNG\_RingOscDiv2 = 1U,  
    kTRNG\_RingOscDiv4 = 2U,  
    kTRNG\_RingOscDiv8 = 3U }  
*TRNG ring oscillator divide.*

## Functions

- status\_t `TRNG_GetDefaultConfig` (`trng_config_t` \*userConfig)  
*Initializes user configuration structure to default.*
- status\_t `TRNG_Init` (`TRNG_Type` \*base, const `trng_config_t` \*userConfig)  
*Initializes the TRNG.*
- void `TRNG_Deinit` (`TRNG_Type` \*base)  
*Shuts down the TRNG.*
- status\_t `TRNG_GetRandomData` (`TRNG_Type` \*base, void \*data, `size_t` dataSize)  
*Gets random data.*

## Driver version

- #define `FSL_TRNG_DRIVER_VERSION` (`MAKE_VERSION`(2, 0, 1))

*TRNG driver version 2.0.1.*

## 35.4 Data Structure Documentation

### 35.4.1 struct trng\_statistical\_check\_limit\_t

Used by [trng\\_config\\_t](#).

#### Data Fields

- `uint32_t maximum`  
*Maximum limit.*
- `uint32_t minimum`  
*Minimum limit.*

#### 35.4.1.0.0.20 Field Documentation

##### 35.4.1.0.0.20.1 `uint32_t trng_statistical_check_limit_t::maximum`

##### 35.4.1.0.0.20.2 `uint32_t trng_statistical_check_limit_t::minimum`

### 35.4.2 struct trng\_config\_t

This structure initializes the TRNG by calling the the [TRNG\\_Init\(\)](#) function. It contains all TRNG configurations.

#### Data Fields

- `bool lock`  
*Disable programmability of TRNG registers.*
- `trng_clock_mode_t clockMode`  
*Clock mode used to operate TRNG.*
- `trng_ring_osc_div_t ringOscDiv`  
*Ring oscillator divide used by TRNG.*
- `trng_sample_mode_t sampleMode`  
*Sample mode of the TRNG ring oscillator.*
- `uint16_t entropyDelay`  
*Entropy Delay.*
- `uint16_t sampleSize`  
*Sample Size.*
- `uint16_t sparseBitLimit`  
*Sparse Bit Limit which defines the maximum number of consecutive samples that may be discarded before an error is generated.*
- `uint8_t retryCount`  
*Retry count.*
- `uint8_t longRunMaxLimit`

## Data Structure Documentation

*Largest allowable number of consecutive samples of all 1, or all 0, that is allowed during the Entropy generation.*

- [`trng\_statistical\_check\_limit\_t monobitLimit`](#)

*Maximum and minimum limits for statistical check of number of ones/zero detected during entropy generation.*

- [`trng\_statistical\_check\_limit\_t runBit1Limit`](#)

*Maximum and minimum limits for statistical check of number of runs of length 1 detected during entropy generation.*

- [`trng\_statistical\_check\_limit\_t runBit2Limit`](#)

*Maximum and minimum limits for statistical check of number of runs of length 2 detected during entropy generation.*

- [`trng\_statistical\_check\_limit\_t runBit3Limit`](#)

*Maximum and minimum limits for statistical check of number of runs of length 3 detected during entropy generation.*

- [`trng\_statistical\_check\_limit\_t runBit4Limit`](#)

*Maximum and minimum limits for statistical check of number of runs of length 4 detected during entropy generation.*

- [`trng\_statistical\_check\_limit\_t runBit5Limit`](#)

*Maximum and minimum limits for statistical check of number of runs of length 5 detected during entropy generation.*

- [`trng\_statistical\_check\_limit\_t runBit6PlusLimit`](#)

*Maximum and minimum limits for statistical check of number of runs of length 6 or more detected during entropy generation.*

- [`trng\_statistical\_check\_limit\_t pokerLimit`](#)

*Maximum and minimum limits for statistical check of "Poker Test".*

- [`trng\_statistical\_check\_limit\_t frequencyCountLimit`](#)

*Maximum and minimum limits for statistical check of entropy sample frequency count.*

### 35.4.2.0.0.21 Field Documentation

#### 35.4.2.0.0.21.1 `bool trng_config_t::lock`

#### 35.4.2.0.0.21.2 `trng_clock_mode_t trng_config_t::clockMode`

#### 35.4.2.0.0.21.3 `trng_ring_osc_div_t trng_config_t::ringOscDiv`

#### 35.4.2.0.0.21.4 `trng_sample_mode_t trng_config_t::sampleMode`

#### 35.4.2.0.0.21.5 `uint16_t trng_config_t::entropyDelay`

Defines the length (in system clocks) of each Entropy sample taken.

#### 35.4.2.0.0.21.6 `uint16_t trng_config_t::sampleSize`

Defines the total number of Entropy samples that will be taken during Entropy generation.

#### 35.4.2.0.0.21.7 `uint16_t trng_config_t::sparseBitLimit`

This limit is used only for during von Neumann sampling (enabled by `TRNG_HAL_SetSampleMode()`). Samples are discarded if two consecutive raw samples are both 0 or both 1. If this discarding occurs for a

long period of time, it indicates that there is insufficient Entropy.

#### 35.4.2.0.0.21.8 `uint8_t trng_config_t::retryCount`

It defines the number of times a statistical check may fails during the TRNG Entropy Generation before generating an error.

#### 35.4.2.0.0.21.9 `uint8_t trng_config_t::longRunMaxLimit`

#### 35.4.2.0.0.21.10 `trng_statistical_check_limit_t trng_config_t::monobitLimit`

#### 35.4.2.0.0.21.11 `trng_statistical_check_limit_t trng_config_t::runBit1Limit`

#### 35.4.2.0.0.21.12 `trng_statistical_check_limit_t trng_config_t::runBit2Limit`

#### 35.4.2.0.0.21.13 `trng_statistical_check_limit_t trng_config_t::runBit3Limit`

#### 35.4.2.0.0.21.14 `trng_statistical_check_limit_t trng_config_t::runBit4Limit`

#### 35.4.2.0.0.21.15 `trng_statistical_check_limit_t trng_config_t::runBit5Limit`

#### 35.4.2.0.0.21.16 `trng_statistical_check_limit_t trng_config_t::runBit6PlusLimit`

#### 35.4.2.0.0.21.17 `trng_statistical_check_limit_t trng_config_t::pokerLimit`

#### 35.4.2.0.0.21.18 `trng_statistical_check_limit_t trng_config_t::frequencyCountLimit`

## 35.5 Macro Definition Documentation

### 35.5.1 `#define FSL_TRNG_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))`

Current version: 2.0.1

Change log:

- Version 2.0.1
  - add support for KL8x and KL28Z
  - update default OSCDIV for K81 to divide by 2

## 35.6 Enumeration Type Documentation

### 35.6.1 `enum trng_sample_mode_t`

Used by [trng\\_config\\_t](#).

Enumerator

***kTRNG\_SampleModeVonNeumann*** Use von Neumann data in both Entropy shifter and Statistical Checker.

***kTRNG\_SampleModeRaw*** Use raw data into both Entropy shifter and Statistical Checker.

## Function Documentation

***kTRNG\_SampleModeVonNeumannRaw*** Use von Neumann data in Entropy shifter. Use raw data into Statistical Checker.

### 35.6.2 enum trng\_clock\_mode\_t

Used by [trng\\_config\\_t](#).

Enumerator

***kTRNG\_ClockModeRingOscillator*** Ring oscillator is used to operate the TRNG (default).

***kTRNG\_ClockModeSystem*** System clock is used to operate the TRNG. This is for test use only, and indeterminate results may occur.

### 35.6.3 enum trng\_ring\_osc\_div\_t

Used by [trng\\_config\\_t](#).

Enumerator

***kTRNG\_RingOscDiv0*** Ring oscillator with no divide.

***kTRNG\_RingOscDiv2*** Ring oscillator divided-by-2.

***kTRNG\_RingOscDiv4*** Ring oscillator divided-by-4.

***kTRNG\_RingOscDiv8*** Ring oscillator divided-by-8.

## 35.7 Function Documentation

### 35.7.1 status\_t TRNG\_GetDefaultConfig ( [trng\\_config\\_t](#) \* *userConfig* )

This function initializes the configure structure to default value. the default value are:

```
* user_config->lock = 0;
* user_config->clockMode = kTRNG\_ClockModeRingOscillator;
* user_config->ringOscDiv = kTRNG\_RingOscDiv0; Or to other kTRNG_RingOscDiv[2|8]
depending on platform.
* user_config->sampleMode = kTRNG\_SampleModeRaw;
* user_config->entropyDelay = 3200;
* user_config->sampleSize = 2500;
* user_config->sparseBitLimit = TRNG_USER_CONFIG_DEFAULT_SPARSE_BIT_LIMIT;
* user_config->retryCount = 63;
* user_config->longRunMaxLimit = 34;
* user_config->monobitLimit.maximum = 1384;
* user_config->monobitLimit.minimum = 1116;
* user_config->runBit1Limit.maximum = 405;
* user_config->runBit1Limit.minimum = 227;
* user_config->runBit2Limit.maximum = 220;
* user_config->runBit2Limit.minimum = 98;
* user_config->runBit3Limit.maximum = 125;
* user_config->runBit3Limit.minimum = 37;
* user_config->runBit4Limit.maximum = 75;
```

```

*   user_config->runBit4Limit.minimum = 11;
*   user_config->runBit5Limit.maximum = 47;
*   user_config->runBit5Limit.minimum = 1;
*   user_config->runBit6PlusLimit.maximum = 47;
*   user_config->runBit6PlusLimit.minimum = 1;
*   user_config->pokerLimit.maximum = 26912;
*   user_config->pokerLimit.minimum = 24445;
*   user_config->frequencyCountLimit.maximum = 25600;
*   user_config->frequencyCountLimit.minimum = 1600;
*

```

#### Parameters

|                    |                               |
|--------------------|-------------------------------|
| <i>user_config</i> | User configuration structure. |
|--------------------|-------------------------------|

#### Returns

If successful, returns the kStatus\_TRNG\_Success. Otherwise, it returns an error.

### 35.7.2 **status\_t TRNG\_Init ( TRNG\_Type \* *base*, const trng\_config\_t \* *userConfig* )**

This function initializes the TRNG. When called, the TRNG entropy generation starts immediately.

#### Parameters

|                   |                                                |
|-------------------|------------------------------------------------|
| <i>base</i>       | TRNG base address                              |
| <i>userConfig</i> | Pointer to initialize configuration structure. |

#### Returns

If successful, returns the kStatus\_TRNG\_Success. Otherwise, it returns an error.

### 35.7.3 **void TRNG\_Deinit ( TRNG\_Type \* *base* )**

This function shuts down the TRNG.

#### Parameters

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

### 35.7.4 **status\_t TRNG\_GetRandomData ( TRNG\_Type \* *base*, void \* *data*, size\_t *dataSize* )**

This function gets random data from the TRNG.

## Function Documentation

Parameters

|                 |                                                  |
|-----------------|--------------------------------------------------|
| <i>base</i>     | TRNG base address                                |
| <i>data</i>     | Pointer address used to store random data        |
| <i>dataSize</i> | Size of the buffer pointed by the data parameter |

Returns

random data

# Chapter 36

## VREF: Voltage Reference Driver

### 36.1 Overview

The KSDK provides a peripheral driver for the Crossbar Voltage Reference (VREF) block of Kinetis devices.

The Voltage Reference(VREF) is intended to supply 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.

To configure the VREF driver, configure `vref_config_t` structure in one of two ways.

1. Use the `VREF_GetDefaultConfig()` function.
2. Sets the parameter in `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.

### 36.2 Typical use case and example

This example shows how to generate a reference voltage by using the VREF module.

```
vref_config_t vrefUserConfig;
VREF_GetDefaultConfig(&vrefUserConfig); /* Gets a default configuration. */
VREF_Init(VREF, &vrefUserConfig);      /* Initializes and configures the VREF module */

/* Do something */

VREF_Deinit(VREF); /* De-initializes the VREF module */
```

## Data Structures

- struct `vref_config_t`  
*The description structure for the VREF module. [More...](#)*

## Enumerations

- enum `vref_buffer_mode_t`{  
    `kVREF_ModeBandgapOnly` = 0U,  
    `kVREF_ModeTightRegulationBuffer` = 2U }  
*VREF modes.*

## Driver version

- #define `FSL_VREF_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 0)`)  
*Version 2.1.0.*

## Function Documentation

### VREF functional operation

- void **VREF\_Init** (VREF\_Type \*base, const vref\_config\_t \*config)  
*Enables the clock gate and configures the VREF module according to the configuration structure.*
- void **VREF\_Deinit** (VREF\_Type \*base)  
*Stops and disables the clock for the VREF module.*
- void **VREF\_GetDefaultConfig** (vref\_config\_t \*config)  
*Initializes the VREF configuration structure.*
- void **VREF\_SetTrimVal** (VREF\_Type \*base, uint8\_t trimValue)  
*Sets a TRIM value for reference voltage.*
- static uint8\_t **VREF\_GetTrimVal** (VREF\_Type \*base)  
*Reads the value of the TRIM meaning output voltage.*

### 36.3 Data Structure Documentation

#### 36.3.1 struct vref\_config\_t

##### Data Fields

- vref\_buffer\_mode\_t bufferMode  
*Buffer mode selection.*

### 36.4 Macro Definition Documentation

#### 36.4.1 #define FSL\_VREF\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 0))

### 36.5 Enumeration Type Documentation

#### 36.5.1 enum vref\_buffer\_mode\_t

Enumerator

**kVREF\_ModeBandgapOnly** Bandgap on only, for stabilization and startup.

**kVREF\_ModeTightRegulationBuffer** Tight regulation buffer enabled.

### 36.6 Function Documentation

#### 36.6.1 void VREF\_Init ( VREF\_Type \* *base*, const vref\_config\_t \* *config* )

This function must be called before calling all the other VREF driver functions, read/write registers, and configurations with user-defined settings. The example below shows how to set up **vref\_config\_t** parameters and how to call the **VREF\_Init** function by passing in these parameters: 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. |

### 36.6.2 void VREF\_Deinit ( VREF\_Type \* *base* )

This function should be called to shut down the module. Example:

```
*     vref_config_t vrefUserConfig;
*     VREF_Init(VREF);
*     VREF_GetDefaultConfig(&vrefUserConfig);
*     ...
*     VREF_Deinit(VREF);
*
```

Parameters

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

### 36.6.3 void VREF\_GetDefaultConfig ( vref\_config\_t \* *config* )

This function initializes the VREF configuration structure to a default value. Example:

```
*     vrefConfig->bufferMode = kVREF_ModeHighPowerBuffer;
*     vrefConfig->enableExternalVoltRef = false;
*     vrefConfig->enableLowRef = false;
*
```

Parameters

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

### 36.6.4 void VREF\_SetTrimVal ( VREF\_Type \* *base*, uint8\_t *trimValue* )

This function sets a TRIM value for reference voltage. Note that the TRIM value maximum is 0x3F.

## Function Documentation

Parameters

|                  |                                                                                        |
|------------------|----------------------------------------------------------------------------------------|
| <i>base</i>      | VREF peripheral address.                                                               |
| <i>trimValue</i> | Value of the trim register to set the output reference voltage (maximum 0x3F (6-bit)). |

### 36.6.5 static uint8\_t VREF\_GetTrimVal ( VREF\_Type \* *base* ) [inline], [static]

This function gets the TRIM value from the TRM register.

Parameters

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

Returns

Six-bit value of trim setting.

# Chapter 37

## WDOG: Watchdog Timer Driver

### 37.1 Overview

The KSDK provides a peripheral driver for the Watchdog module (WDOG) of Kinetis devices.

### 37.2 Typical use case

```
wdog_config_t config;  
WDOG_GetDefaultConfig(&config);  
config.timeoutValue = 0x7ffU;  
config.enableWindowMode = true;  
config.windowValue = 0x1ffU;  
WDOG_Init(wdog_base, &config);
```

## Data Structures

- struct `wdog_work_mode_t`  
*Defines WDOG work mode. [More...](#)*
- struct `wdog_config_t`  
*Describes WDOG configuration structure. [More...](#)*
- struct `wdog_test_config_t`  
*Describes WDOG test mode configuration structure. [More...](#)*

## Enumerations

- enum `wdog_clock_source_t` {  
    kWDOG\_LpoClockSource = 0U,  
    kWDOG\_AlternateClockSource = 1U }  
*Describes WDOG clock source.*
- enum `wdog_clock_prescaler_t` {  
    kWDOG\_ClockPrescalerDivide1 = 0x0U,  
    kWDOG\_ClockPrescalerDivide2 = 0x1U,  
    kWDOG\_ClockPrescalerDivide3 = 0x2U,  
    kWDOG\_ClockPrescalerDivide4 = 0x3U,  
    kWDOG\_ClockPrescalerDivide5 = 0x4U,  
    kWDOG\_ClockPrescalerDivide6 = 0x5U,  
    kWDOG\_ClockPrescalerDivide7 = 0x6U,  
    kWDOG\_ClockPrescalerDivide8 = 0x7U }  
*Describes the selection of the clock prescaler.*
- enum `wdog_test_mode_t` {  
    kWDOG\_QuickTest = 0U,  
    kWDOG\_ByteTest = 1U }  
*Describes WDOG test mode.*

## Typical use case

- 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, 0)`)  
    *Defines WDOG driver version 2.0.0.*

## 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 WDOG configure sturcture.*
- 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 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)  
*Enable WDOG interrupt.*
- static void [WDOG\\_DisableInterrupts](#) (WDOG\_Type \*base, uint32\_t mask)  
*Disable WDOG interrupt.*
- uint32\_t [WDOG\\_GetStatusFlags](#) (WDOG\_Type \*base)  
*Gets WDOG all status flags.*
- void [WDOG\\_ClearStatusFlags](#) (WDOG\_Type \*base, uint32\_t mask)  
*Clear WDOG flag.*
- static void [WDOG\\_SetTimeoutValue](#) (WDOG\_Type \*base, uint32\_t timeoutCount)  
*Set 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.*

## 37.3 Data Structure Documentation

### 37.3.1 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.*

### 37.3.2 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](#)

## Enumeration Type Documentation

- `bool enableWindowMode`  
*Enables or disables WDOG interrupt.*
- `uint32_t windowValue`  
*Window value.*
- `uint32_t timeoutValue`  
*Timeout value.*

### 37.3.3 `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.*

## 37.4 Macro Definition Documentation

### 37.4.1 `#define FSL_WDOG_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

## 37.5 Enumeration Type Documentation

### 37.5.1 `enum wdog_clock_source_t`

Enumerator

`kWDOG_LpoClockSource` WDOG clock sourced from LPO.

`kWDOG_AlternateClockSource` WDOG clock sourced from alternate clock source.

### 37.5.2 `enum wdog_clock_prescaler_t`

Enumerator

`kWDOG_ClockPrescalerDivide1` Divided by 1.

`kWDOG_ClockPrescalerDivide2` Divided by 2.

`kWDOG_ClockPrescalerDivide3` Divided by 3.

`kWDOG_ClockPrescalerDivide4` Divided by 4.

`kWDOG_ClockPrescalerDivide5` Divided by 5.

`kWDOG_ClockPrescalerDivide6` Divided by 6.

`kWDOG_ClockPrescalerDivide7` Divided by 7.

`kWDOG_ClockPrescalerDivide8` Divided by 8.

### 37.5.3 enum wdog\_test\_mode\_t

Enumerator

*kWDOG\_QuickTest* Selects quick test.

*kWDOG\_Bytetest* Selects byte test.

### 37.5.4 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.

### 37.5.5 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.

### 37.5.6 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.

## 37.6 Function Documentation

### 37.6.1 void WDOG\_GetDefaultConfig ( wdog\_config\_t \* config )

This function initializes the WDOG configuration structure to default value. The default values are:

```
*   wdogConfig->enableWdog = true;
*   wdogConfig->clockSource = kWDOG_IpoClockSource;
*   wdogConfig->prescaler = kWDOG_ClockPrescalerDivide1;
*   wdogConfig->workMode.enableWait = true;
```

## Function Documentation

```
* 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 WDOG config structure. |
|---------------|-----------------------------------|

### See Also

[wdog\\_config\\_t](#)

### 37.6.2 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. If user wants to reconfigure WDOG without forcing a reset first, enableUpdate must be set to true in configuration.

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    |

### 37.6.3 void WDOG\_Deinit ( **WDOG\_Type** \* *base* )

This function shuts down the WDOG. Make sure that the WDOG\_STCTRLH.ALLOWUPDATE is 1 which means that the register update is enabled.

### 37.6.4 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. Make sure that the WDOG\_STCTRLH.ALLOWUPDATE is 1 which means that the register update is enabled.

Example:

```
*     wdog_test_config_t test_config;
*     test_config.testMode = kWDOG_QuickTest;
*     test_config.timeoutValue = 0xfffffffu;
*     WDOG_SetTestModeConfig(wdog_base, &test_config);
*
```

Parameters

|               |                                           |
|---------------|-------------------------------------------|
| <i>base</i>   | WDOG peripheral base address              |
| <i>config</i> | The functional test configuration of WDOG |

### 37.6.5 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 |
|-------------|------------------------------|

### 37.6.6 static void WDOG\_Disable( WDOG\_Type \* *base* ) [inline], [static]

This function write value into WDOG\_STCTRLH register to disable 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 |
|-------------|------------------------------|

### 37.6.7 static void WDOG\_EnableInterrupts( WDOG\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function write value into WDOG\_STCTRLH register to enable WDOG interrupt, 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.

## Function Documentation

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> |

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

This function write value into WDOG\_STCTRLH register to disable WDOG interrupt, 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                                                                                                                                           |
| <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> |

### 37.6.9 **uint32\_t** WDOG\_GetStatusFlags ( **WDOG\_Type** \* *base* )

This function gets all status flags.

Example for getting 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.

### 37.6.10 void WDOG\_ClearStatusFlags ( **WDOG\_Type** \* *base*, **uint32\_t** *mask* )

This function clears WDOG status flag.

Example for clearing 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 |

### 37.6.11 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 write value into WDOG\_TOVALH and WDOG\_TOVALL registers which are wirte-once. Make sure the WCT window is still open and these two registers have not been written in this WCT while this function is called.

Parameters

|                     |                                               |
|---------------------|-----------------------------------------------|
| <i>base</i>         | WDOG peripheral base address                  |
| <i>timeoutCount</i> | WDOG timeout value, count of WDOG clock tick. |

### 37.6.12 static void WDOG\_SetWindowValue ( **WDOG\_Type** \* *base*, **uint32\_t** *windowValue* ) [inline], [static]

This function sets the WDOG window value. This function write value into WDOG\_WINH and WDOG\_WINL registers which are wirte-once. Make sure the WCT window is still open and these two registers have not been written in this WCT while this function is called.

## Function Documentation

Parameters

|                    |                              |
|--------------------|------------------------------|
| <i>base</i>        | WDOG peripheral base address |
| <i>windowValue</i> | WDOG window value.           |

### 37.6.13 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 could effectively invalidate the unlock sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

Parameters

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

### 37.6.14 void WDOG\_Refresh ( WDOG\_Type \* *base* )

This function feeds the WDOG. This function should be called before WDOG timer is in timeout. Otherwise, a reset is asserted.

Parameters

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

### 37.6.15 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

**37.6.16 static void WDOG\_ClearResetCount( WDOG\_Type \* *base* ) [inline],  
[static]**

This function clears the WDOG reset count value.

## Function Documentation

### Parameters

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

# Chapter 38

## Debug Console

### 38.1 Overview

This part 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.

### 38.2 Function groups

#### 38.2.1 Initialization

To initialize the debug console, call the DbgConsole\_Init() function with these parameters. This function automatically enables the module and the clock.

```
/*
 * @brief Initializes the the peripheral used to debug messages.
 *
 * @param baseAddr      Indicates which address of the peripheral is used to send debug messages.
 * @param baudRate     The desired baud rate in bits per second.
 * @param device        Low level device type for the debug console, can be one of:
 *                      @arg DEBUG_CONSOLE_DEVICE_TYPE_UART,
 *                      @arg DEBUG_CONSOLE_DEVICE_TYPE_LPUART,
 *                      @arg DEBUG_CONSOLE_DEVICE_TYPE_LPSCI,
 *                      @arg DEBUG_CONSOLE_DEVICE_TYPE_USBCDC.
 * @param clkSrcFreq   Frequency of peripheral source clock.
 *
 * @return              Whether initialization was successful or not.
 */
status_t DbgConsole_Init(uint32_t baseAddr, uint32_t baudRate, uint8_t device, uint32_t clkSrcFreq)
```

Selects the supported debug console hardware device type, such as

```
DEBUG_CONSOLE_DEVICE_TYPE_NONE
DEBUG_CONSOLE_DEVICE_TYPE_LPSCI
DEBUG_CONSOLE_DEVICE_TYPE_UART
DEBUG_CONSOLE_DEVICE_TYPE_LPUART
DEBUG_CONSOLE_DEVICE_TYPE_USBCDC
```

After the initialization is successful, stdout and stdin are connected to the selected peripheral. The debug console state is stored in the debug\_console\_state\_t structure, such as shown here:

```
typedef struct DebugConsoleState
{
    uint8_t                  type;
    void*                   base;
    debug_console_ops_t     ops;
} debug_console_state_t;
```

## Function groups

This example shows how to call the DbgConsole\_Init() given the user configuration structure:

```
uint32_t uartClkSrcFreq = CLOCK_GetFreq(BOARD_DEBUG_UART_CLKSRC);  
  
DbgConsole_Init(BOARD_DEBUG_UART_BASEADDR, BOARD_DEBUG_UART_BAUDRATE, DEBUG_CONSOLE_DEVICE_TYPE_UART,  
                 uartClkSrcFreq);
```

### 38.2.2 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 going to be 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                              |

## Function groups

- 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, i.e., 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 *           |

| specifier              | Qualifying Input                                                                                                                                                                                                            | Type of argument |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| 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(const char *fmt_ptr, ...);
int DbgConsole_Getchar(void);
```

This utility supports selecting toolchain's printf/scanf or the KSDK printf/scanf:

```
#if SDK_DEBUGCONSOLE      /* Select printf, scanf, putchar, getchar of SDK version. */
#define PRINTF            DbgConsole_Printf
#define SCANF             DbgConsole_Scanf
#define PUTCHAR           DbgConsole_Putchar
#define GETCHAR           DbgConsole_Getchar
#else                      /* Select printf, scanf, putchar, getchar of toolchain. */
#define PRINTF            printf
#define SCANF             scanf
#define PUTCHAR           putchar
#define GETCHAR           getchar
#endif /* SDK_DEBUGCONSOLE */
```

### 38.3 Typical use case

#### Some examples use the PUTCHAR & GETCHAR function

```
ch = GETCHAR();
PUTCHAR(ch);
```

## Typical use case

### 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\r\nTime: %u ticks %2.5f milliseconds\r\nDONE\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 KSDK \_\_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:

If you want to use 'printf' and 'scanf' for GNUC Base, you should add file '**fsl\_sbrk.c**' in path-  
: ..\{package}\devices\{subset}\utilities\fsl\_sbrk.c to your project.

## Modules

- Semihosting

## 38.4 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 could 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

### 38.4.1 Guide Semihosting for IAR

**NOTE:** After the setting both "printf" and "scanf" are available for debugging

#### 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 will ensure that the debug session will start by running to 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. Start the project by choosing Project>Download and Debug.
4. Choose View>Terminal I/O to display the output from the I/O operations.

### 38.4.2 Guide Semihosting for Keil µVision

**NOTE:** Keil supports Semihosting only for M3/M4 cores.

#### Step 1: Prepare code

Remove function `fputc` and `fgetc` is used to support KEIL in "fsl\_debug\_console.c" then add the following code to project:

```
#pragma import(__use_no_semihosting_swi)

volatile int ITM_RxBuffer = ITM_RXBUFFER_EMPTY;           /* used for Debug Input */
```

## Semihosting

```
struct __FILE
{
    int handle;
};

FILE __stdout;
FILE __stdin;

int fputc(int ch, FILE *f)
{
    return (ITM_SendChar(ch));
}

int fgetc(FILE *f)
{ /* blocking */
    while (ITM_CheckChar() != 1)
        ;
    return (ITM_ReceiveChar());
}

int ferror(FILE *f)
{
    /* Your implementation of ferror */
    return EOF;
}

void _ttywrch(int ch)
{
    ITM_SendChar(ch);
}

void _sys_exit(int return_code)
{
label:
    goto label; /* endless loop */
}
```

### Step 2: Setting up the environment

1. In menu bar, choose Project>Options for target or using Alt+F7 or click
2. Next, select "Target" tab and not select "Use MicroLIB".
3. Next, select "Debug" tab, select "J-LINK/J-TRACE Cortex" and click "Setting button".
4. Next, select "Debug" tab and choose Port:SW, then select "Trace" tab, choose "Enable" and click OK

### Step 3: Building the project

1. Compile and link the project by choosing Project>Build Target or using F7

### Step 4: Building the project

1. Choose "Debug" on menu bar or Ctrl F5
2. In menu bar, choose "Serial Window" and click to "Debug (printf) Viewer"
3. Run line by line to see result in Console Window.

### 38.4.3 Guide Semihosting for KDS

**NOTE:** After the setting we can use "printf" for debugging

#### Step 1: Setting up the environment

1. In menu bar, choose Project>Properties>C/C++ Build>Settings>Tool Settings.
2. Select “Libraries” on “Cross ARM C Linker” and delete “nosys”.
3. Select “Miscellaneous” on “Cross ARM C Linker”, add “-specs=rdimon.specs” to “Other link flags” and tick “Use newlib-nano” and click OK.

#### Step 2: Building the project

1. In menu bar, choose Project>Build Project.

#### Step 3: Starting semihosting

1. In Debug configurations, choose "Startup" tab, tick “Enable semihosting and Telnet”. Press “Apply” and “Debug”.
2. After click Debug, the Window same as below, run line by line to see result in Console Window.

### 38.4.4 Guide Semihosting for ATL

**NOTE:** Hardware jlink have to be used to enable semihosting

#### Step 1: Prepare code

Add the following code to project:

```
int _write(int file, char *ptr, int len)
{
    /* Implement your write code here, this is used by puts and printf for example */
    int i=0;
    for(i=0 ; i<len ; i++)
        ITM_SendChar((*ptr++));
    return len;
}
```

#### Step 2: Setting up the environment

1. In menu bar, choose Debug Configurations. In tab "Embedded C/C++ Application" choose "-Semihosting\_ATL\_xxx debug jlink".
2. In tab "Debugger" setup like that:
  - JTAG mode must be selected
  - SWV tracing must be enabled

- Enter the Core Clock frequency. This is H/W board specific.
  - Enter the desired SWO Clock frequency. The latter depends on the JTAG Probe and must be a multiple of the Core Clock value.
3. Click "Apply" and "Debug".

### Step 3: Starting semihosting

1. In the Views menu, expand the submenu SWV and open the docking view "SWV Console".
2. Open the SWV settings panel by clicking on the Configure Serial Wire Viewer button in the SWV Console view toolbar.
3. Configure the data ports to be traced by enabling the ITM channel 0 check-box in the ITM stimulus ports group: Choose "EXETRC: Trace Exceptions" and In tab "ITM Stimulus Ports" choose "Enable Port" 0. Then click "OK".
4. Recommend not enabling other SWV trace functionalities at the same time, as this may over-use the SWO pin causing packet loss due to limited bandwidth (certain other SWV tracing capabilities can send a lot of data at very high speed). Save the SWV configuration by clicking the OK button. The configuration is saved together with other debug configurations and will remain effective until changed.
5. Press the red Start/Stop Trace button to send the SWV configuration to the target board and enable SWV trace recording. The board will not send any SWV packages until it is properly configured. The SWV Configuration must be resent, if the configuration registers on the target board are reset. Also, actual tracing will not start until the target starts to execute
6. Start the target execution again by pressing the green Resume Debug button.
7. The SWV console will now show the printf() output

### 38.4.5 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". Setup like this :
  - "Host Name (or IP address)" : localhost
  - "Port" :2333
  - "Connection type" : Telnet.
  - 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}
```

## Semihosting

```
G} --specs=rdimon.specs ")  
Remove  
target_link_libraries(semihosting_ARMGCC.elf debug nosys)  
2. Run "build_debug.bat" to build project
```

### Step 3: Starting semihosting

- (a) Download the image and set like this:

```
cd D:\mcu-sdk-2.0-origin\boards\twrk64f120m\driver_examples\semihosting\armgcc\debug  
d:  
C:\PROGRA~2\GNUTOO~1\4BD65~1.920\bin\arm-none-eabi-gdb.exe  
target remote localhost:2331  
monitor reset  
monitor semihosting enable  
monitor semihosting thumbSWI 0xAB  
monitor semihosting IOClient 1  
monitor flash device = MK64FN1M0xxx12  
load semihosting_ARMGCC.elf  
monitor reg pc = (0x00000004)  
monitor reg sp = (0x00000000)  
continue
```

- (b) After the setting, press "enter", the PuTTY window will now show the printf() output.

# Chapter 39

## Notification Framework

### 39.1 Overview

This section describes the programming interface of the Notifier driver.

### 39.2 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.

The configuration transition includes 3 steps:

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 "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 changes to the new configuration.
3. After the configuration changes, the Notifier sends an "AFTER" message to the callback table to notify drivers that the configuration transition is finished.

This example shows how to use the Notifier in the Power Manager application:

```
#include "fsl_notifier.h"

/* Definition of the Power Manager callback */
status_t callback0(notifier_notification_block_t *notify, void *data)
{
    status_t ret = kStatus_Success;

    ...
    ...

    return ret;
}
/* Definition of the Power Manager user function */
status_t APP_PowerModeSwitch(notifier_user_config_t *targetConfig, void *userData)
```

## Notifier Overview

# 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, indicates what 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)`  
*Create Notifier handle.*
- `status_t NOTIFIER_SwitchConfig (notifier_handle_t *notifierHandle, uint8_t configIndex, notifier_policy_t policy)`  
*Switch configuration according to a pre-defined structure.*
- `uint8_t NOTIFIER_GetErrorCallbackIndex (notifier_handle_t *notifierHandle)`  
*This function returns the last failed notification callback.*

## 39.3 Data Structure Documentation

### 39.3.1 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.*

## Data Structure Documentation

### 39.3.1.0.0.22 Field Documentation

39.3.1.0.0.22.1 `notifier_user_config_t* notifier_notification_block_t::targetConfig`

39.3.1.0.0.22.2 `notifier_policy_t notifier_notification_block_t::policy`

39.3.1.0.0.22.3 `notifier_notification_type_t notifier_notification_block_t::notifyType`

### 39.3.2 `struct notifier_callback_config_t`

This structure holds configuration of callbacks. Callbacks of this type are expected to be statically allocated. This structure contains 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.*

### 39.3.2.0.0.23 Field Documentation

39.3.2.0.0.23.1 `notifier_callback_t notifier_callback_config_t::callback`

39.3.2.0.0.23.2 `notifier_callback_type_t notifier_callback_config_t::callbackType`

39.3.2.0.0.23.3 `void* notifier_callback_config_t::callbackData`

### 39.3.3 `struct notifier_handle_t`

Notifier handle structure. Contains data necessary for 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.*

### 39.3.3.0.0.24 Field Documentation

**39.3.3.0.0.24.1 `notifier_user_config_t** notifier_handle_t::configsTable`**

**39.3.3.0.0.24.2 `uint8_t notifier_handle_t::configsNumber`**

**39.3.3.0.0.24.3 `notifier_callback_config_t* notifier_handle_t::callbacksTable`**

**39.3.3.0.0.24.4 `uint8_t notifier_handle_t::callbacksNumber`**

**39.3.3.0.0.24.5 `uint8_t notifier_handle_t::errorCallbackIndex`**

**39.3.3.0.0.24.6 `uint8_t notifier_handle_t::currentConfigIndex`**

**39.3.3.0.0.24.7 `notifier_user_function_t notifier_handle_t::userFunction`**

**39.3.3.0.0.24.8 `void* notifier_handle_t::userData`**

## 39.4 Typedef Documentation

### 39.4.1 **typedef void notifier\_user\_config\_t**

Reference of user defined configuration is stored in an array; the notifier switches between these configurations based on this array.

### 39.4.2 **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

---

## Enumeration Type Documentation

|                     |                                                        |
|---------------------|--------------------------------------------------------|
| <i>targetConfig</i> | target Configuration.                                  |
| <i>userData</i>     | Refers to other specific data passed to user function. |

Returns

An error code or kStatus\_Success.

### 39.4.3 **typedef status\_t(\* notifier\_callback\_t)(notifier\_notification\_block\_t \*notify, void \*data)**

Declaration of callback. It is common for registered callbacks. Reference to function of this type is part of [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, type of the notification is passed as parameter along with reference to the target configuration structure (see [notifier\\_notification\\_block\\_t](#)) and any data passed during the callback registration. When notified before configuration switch, depending on the configuration switch policy (see [notifier\\_policy\\_t](#)) the callback may deny the execution of user function by returning any error code different from 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.

## 39.5 Enumeration Type Documentation

### 39.5.1 enum \_notifier\_status

Used as return value of Notifier functions.

Enumerator

***kStatus\_NOTIFIER\_ErrorNotificationBefore*** Error occurs during send "BEFORE" notification.  
***kStatus\_NOTIFIER\_ErrorNotificationAfter*** Error occurs during send "AFTER" notification.

### 39.5.2 enum notifier\_policy\_t

Defines whether 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** user function is executed regardless of the results.

### 39.5.3 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.

### 39.5.4 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 [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 unsuccessful attempt to switch configuration
- after successful configuration switch

Enumerator

**kNOTIFIER\_CallbackBefore** Callback handles BEFORE notification.

**kNOTIFIER\_CallbackAfter** Callback handles AFTER notification.

**kNOTIFIER\_CallbackBeforeAfter** Callback handles BEFORE and AFTER notification.

## Function Documentation

### 39.6 Function Documentation

39.6.1 `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 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 callbacks array.                                                                                |
| <i>userFunction</i>     | user function.                                                                                                                          |
| <i>userData</i>         | user data passed to user function.                                                                                                      |

## Returns

An error code or kStatus\_Success.

### 39.6.2 **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 agreement is required, if any callback returns an error code then 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 returned an error code, an error code denoting in which phase the error occurred is returned when [NOTIFIER\\_SwitchConfig\(\)](#) exits.

## Parameters

## Function Documentation

|                       |                                                                            |
|-----------------------|----------------------------------------------------------------------------|
| <i>notifierHandle</i> | pointer to notifier handle                                                 |
| <i>configIndex</i>    | Index of the target configuration.                                         |
| <i>policy</i>         | Transaction policy, kNOTIFIER_PolicyAgreement or kNOTIFIER_PolicyForcible. |

Returns

An error code or kStatus\_Success.

### 39.6.3 `uint8_t NOTIFIER_GetErrorCallbackIndex ( notifier_handle_t *notifierHandle )`

This function returns index of the last callback that failed during the configuration switch while the last [N-OTIFIER\\_SwitchConfig\(\)](#) was called. If the last [NOTIFIER\\_SwitchConfig\(\)](#) call ended successfully value equal to callbacks number is returned. Returned value represents index in the array of static call-backs.

Parameters

|                       |                            |
|-----------------------|----------------------------|
| <i>notifierHandle</i> | pointer to notifier handle |
|-----------------------|----------------------------|

Returns

Callback index of last failed callback or value equal to callbacks count.

# Chapter 40

## Shell

### 40.1 Overview

This part describes the programming interface of the Shell middleware. Shell controls MCUs by commands via the specified communication peripheral based on the debug console driver.

### 40.2 Function groups

#### 40.2.1 Initialization

To initialize the Shell middleware, call the `SHELL_Init()` function with these parameters. This function automatically enables the middleware.

```
void SHELL_Init(p_shell_context_t context, send_data_cb_t send_cb,  
                 recv_data_cb_t recv_cb, 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(&user_context, SHELL_SendDataCallback, SHELL_ReceiveDataCallback, "SHELL>> ");
```

#### 40.2.2 Advanced Feature

- Support to get a character from standard input devices.

```
static uint8_t GetChar(p_shell_context_t context);
```

| Commands   | Description                                      |
|------------|--------------------------------------------------|
| Help       | Lists all commands which are supported by Shell. |
| Exit       | Exits the Shell program.                         |
| strCompare | Compares the two input strings.                  |

| Input character | Description                                         |
|-----------------|-----------------------------------------------------|
| A               | Gets the latest command in the history.             |
| B               | Gets the first command in the history.              |
| C               | Replaces one character at the right of the pointer. |

## Function groups

| Input character | Description                                        |
|-----------------|----------------------------------------------------|
| D               | Replaces one character at the left of the pointer. |
|                 | Run AutoComplete function                          |
|                 | Run cmdProcess function                            |
|                 | Clears a command.                                  |

### 40.2.3 Shell Operation

```
SHELL_Init(&user_context, SHELL_SendDataCallback, SHELL_ReceiveDataCallback, "SHELL>> ");
SHELL_Main(&user_context);
```

## Data Structures

- struct [p\\_shell\\_context\\_t](#)  
*Data structure for Shell environment.* [More...](#)
- struct [shell\\_command\\_context\\_t](#)  
*User command data structure.* [More...](#)
- struct [shell\\_command\\_context\\_list\\_t](#)  
*Structure list command.* [More...](#)

## Macros

- #define [SHELL\\_USE\\_HISTORY](#) (0U)  
*Macro to set on/off history feature.*
- #define [SHELL\\_SEARCH\\_IN\\_HIST](#) (1U)  
*Macro to set on/off history feature.*
- #define [SHELL\\_USE\\_FILE\\_STREAM](#) (0U)  
*Macro to select method stream.*
- #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\\_HIST\\_MAX](#) (3U)  
*Macro to set maximum count of history commands.*
- #define [SHELL\\_MAX\\_CMD](#) (6U)  
*Macro to set maximum count of commands.*

## Typedefs

- typedef void(\* [send\\_data\\_cb\\_t](#))(uint8\_t \*buf, uint32\_t len)  
*Shell user send data callback prototype.*
- typedef void(\* [recv\\_data\\_cb\\_t](#))(uint8\_t \*buf, uint32\_t len)  
*Shell user receiver data callback prototype.*
- typedef int(\* [printf\\_data\\_t](#))(const char \*format,...)

- *Shell user printf data prototype.*  
**typedef int32\_t(\* cmd\_function\_t )(p\_shell\_context\_t context, int32\_t argc, char \*\*argv)**  
*User command function prototype.*

## Enumerations

- **enum fun\_key\_status\_t {**  
**kSHELL\_Normal = 0U,**  
**kSHELL\_Special = 1U,**  
**kSHELL\_Function = 2U }**  
*A type for the handle special key.*

## Shell functional Operation

- **void SHELL\_Init (p\_shell\_context\_t context, send\_data\_cb\_t send\_cb, recv\_data\_cb\_t recv\_cb, printf\_data\_t shell\_printf, char \*prompt)**  
*Enables the clock gate and configure the Shell module according to the configuration structure.*
- **int32\_t SHELL\_RegisterCommand (const shell\_command\_context\_t \*command\_context)**  
*Shell register command.*
- **int32\_t SHELL\_Main (p\_shell\_context\_t context)**  
*Main loop for Shell.*

## 40.3 Data Structure Documentation

### 40.3.1 struct shell\_context\_struct

#### Data Fields

- **char \* prompt**  
*Prompt string.*
- **enum \_fun\_key\_status stat**  
*Special key status.*
- **char line [SHELL\_BUFFER\_SIZE]**  
*Consult buffer.*
- **uint8\_t cmd\_num**  
*Number of user commands.*
- **uint8\_t l\_pos**  
*Total line position.*
- **uint8\_t c\_pos**  
*Current line position.*
- **send\_data\_cb\_t send\_data\_func**  
*Send data interface operation.*
- **recv\_data\_cb\_t recv\_data\_func**  
*Receive data interface operation.*
- **uint16\_t hist\_current**  
*Current history command in hist buff.*
- **uint16\_t hist\_count**  
*Total history command in hist buff.*
- **char hist\_buf [SHELL\_HIST\_MAX][SHELL\_BUFFER\_SIZE]**

## Data Structure Documentation

- *History buffer.*  
• bool [exit](#)  
*Exit Flag.*

### 40.3.2 struct shell\_command\_context\_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.*

#### 40.3.2.0.0.25 Field Documentation

##### 40.3.2.0.0.25.1 const char\* shell\_command\_context\_t::pcCommand

For example "help". It must be all lower case.

##### 40.3.2.0.0.25.2 char\* shell\_command\_context\_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".

##### 40.3.2.0.0.25.3 const cmd\_function\_t shell\_command\_context\_t::pFuncCallBack

##### 40.3.2.0.0.25.4 uint8\_t shell\_command\_context\_t::cExpectedNumberOfParameters

### 40.3.3 struct shell\_command\_context\_list\_t

#### Data Fields

- const [shell\\_command\\_context\\_t](#) \* [CommandList](#) [[SHELL\\_MAX\\_CMD](#)]  
*The command table list.*
- uint8\_t [numberOfCommandInList](#)  
*The total command in list.*

**40.4 Macro Definition Documentation****40.4.1 #define SHELL\_USE\_HISTORY (0U)****40.4.2 #define SHELL\_SEARCH\_IN\_HIST (1U)****40.4.3 #define SHELL\_USE\_FILE\_STREAM (0U)****40.4.4 #define SHELL\_AUTO\_COMPLETE (1U)****40.4.5 #define SHELL\_BUFFER\_SIZE (64U)****40.4.6 #define SHELL\_MAX\_ARGS (8U)****40.4.7 #define SHELL\_HIST\_MAX (3U)****40.4.8 #define SHELL\_MAX\_CMD (6U)****40.5 Typedef Documentation****40.5.1 typedef void(\* send\_data\_cb\_t)(uint8\_t \*buf, uint32\_t len)****40.5.2 typedef void(\* recv\_data\_cb\_t)(uint8\_t \*buf, uint32\_t len)****40.5.3 typedef int(\* printf\_data\_t)(const char \*format,...)****40.5.4 typedef int32\_t(\* cmd\_function\_t)(p\_shell\_context\_t context, int32\_t argc, char \*\*argv)****40.6 Enumeration Type Documentation****40.6.1 enum fun\_key\_status\_t**

Enumerator

*kSHELL\_Normal* Normal key.*kSHELL\_Special* Special key.*kSHELL\_Function* Function key.

## Function Documentation

### 40.7 Function Documentation

#### 40.7.1 void SHELL\_Init ( *p\_shell\_context\_t context*, *send\_data\_cb\_t send\_cb*, *recv\_data\_cb\_t recv\_cb*, *printf\_data\_t shell\_printf*, *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 middleware Shell and how to call the SHELL\_Init function by passing in these parameters: Example:

```
*     shell_context_struct user_context;
*     SHELL_Init(&user_context, SendDataFunc, ReceiveDataFunc, "SHELL>> ");
*
```

#### Parameters

|                |                                                          |
|----------------|----------------------------------------------------------|
| <i>context</i> | The pointer to the Shell environment and runtime states. |
| <i>send_cb</i> | The pointer to call back send data function.             |
| <i>recv_cb</i> | The pointer to call back receive data function.          |
| <i>prompt</i>  | The string prompt of Shell                               |

#### 40.7.2 int32\_t SHELL\_RegisterCommand ( *const shell\_command\_context\_t \* command\_context* )

#### Parameters

|                        |                                            |
|------------------------|--------------------------------------------|
| <i>command_context</i> | The pointer to the command data structure. |
|------------------------|--------------------------------------------|

#### Returns

-1 if error or 0 if success

#### 40.7.3 int32\_t SHELL\_Main ( *p\_shell\_context\_t context* )

Main loop for Shell; After this function is called, Shell begins to initialize the basic variables and starts to work.

### Parameters

|                |                                                          |
|----------------|----------------------------------------------------------|
| <i>context</i> | The pointer to the Shell environment and runtime states. |
|----------------|----------------------------------------------------------|

### Returns

this function does not return until Shell command exit was called.

## Function Documentation

# Chapter 41

## DMA Manager

### 41.1 Overview

DMA Manager provides a series of functions to manage the DMAMUX channels.

### 41.2 Function groups

#### 41.2.1 DMAMGR Initialization and De-initialization

This function group initializes and deinitializes the DMA Manager.

#### 41.2.2 DMAMGR Operation

This function group requests/releases the DMAMUX channel and configures the channel request source.

### 41.3 Typical use case

#### 41.3.1 DMAMGR static channel allocate

```
DMAMUX_Type *dmamux_base;
uint8_t channel;

/* Initialize DMAMGR */
DMAMGR_Init();
/* Request a DMAMUX channel by static allocate mechanism */
dmamux_base = DMAMUX0;
channel = 0;
DMAMGR_RequestChannel(kDmaRequestMux0AlwaysOn63, &dmamux_base, &channel,
    kDMAMGR_STATIC_ALLOCATE);
```

#### 41.3.2 DMAMGR dynamic channel allocate

```
DMAMUX_Type *dmamux_base;
uint8_t channel;

/* Initialize DMAMGR */
DMAMGR_Init();
/* Request a DMAMUX channel by static allocate mechanism */
dmamux_base = DMAMUX0;
channel = 0;
DMAMGR_RequestChannel(kDmaRequestMux0AlwaysOn63, &dmamux_base, &channel,
    kDMAMGR_DYNAMIC_ALLOCATE);
```

### Macros

- #define **DMAMGR\_DYNAMIC\_ALLOCATE** 0xFFU

## Function Documentation

*Dynamic channel allocate mechanism.*

### Enumerations

- enum `_dma_manager_status` {  
    `kStatus_DMAMGR_ChannelOccupied` = MAKE\_STATUS(kStatusGroup\_DMAMGR, 0),  
    `kStatus_DMAMGR_ChannelNotUsed` = MAKE\_STATUS(kStatusGroup\_DMAMGR, 1),  
    `kStatus_DMAMGR_NoFreeChannel` = MAKE\_STATUS(kStatusGroup\_DMAMGR, 2),  
    `kStatus_DMAMGR_ChannelNotMatchSource` = MAKE\_STATUS(kStatusGroup\_DMAMGR, 3)  
}
- DMA manager status.*

### DMAMGR Initialize and De-initialize

- void `DMAMGR_Init` (void)  
*Initializes the DMA manager.*
- void `DMAMGR_Deinit` (void)  
*Deinitializes the DMA manager.*

### DMAMGR Operation

- status\_t `DMAMGR_RequestChannel` (dma\_request\_source\_t requestSource, uint8\_t virtualChannel, void \*handle)  
*Requests a DMA channel.*
- status\_t `DMAMGR_ReleaseChannel` (void \*handle)  
*Releases a DMA channel.*

## 41.4 Macro Definition Documentation

### 41.4.1 #define DMAMGR\_DYNAMIC\_ALLOCATE 0xFFU

## 41.5 Enumeration Type Documentation

### 41.5.1 enum \_dma\_manager\_status

Enumerator

- `kStatus_DMAMGR_ChannelOccupied` Channel has been occupied.  
`kStatus_DMAMGR_ChannelNotUsed` Channel has not been used.  
`kStatus_DMAMGR_NoFreeChannel` All channel has been occupied.  
`kStatus_DMAMGR_ChannelNotMatchSource` Channel do not match the request source.

## 41.6 Function Documentation

### 41.6.1 void `DMAMGR_Init` ( void )

This function initializes the DMA manager, ungates all DMAMUX clocks, and initializes the eDMA or DMA peripheral.

#### 41.6.2 void DMAMGR\_Deinit ( void )

This function deinitializes the DMA manager, disables all DMAMUX channel, gates all DMAMUX clock, and deinitializes the eDMA or DMA peripheral.

#### 41.6.3 status\_t DMAMGR\_RequestChannel ( *dma\_request\_source\_t requestSource, uint8\_t virtualChannel, void \* handle* )

This function request a DMA channel which is not occupied. There are two channels to allocate the mechanism dynamic and static. For the dynamic allocation mechanism (*virtualChannel* = DMAMGR\_DYNAMIC\_ALLOCATE), DMAMGR allocates a DMA channel according to the given request source and then configure it. For static allocation mechanism, DMAMGR configures the given channel according to the given request source and channel number.

Parameters

|                       |                                                                                                                                                           |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>requestSource</i>  | DMA channel request source number. See the soc.h.                                                                                                         |
| <i>virtualChannel</i> | The channel number user wants to occupy. If using the dynamic channel allocate mechanism, set the <i>virtualChannel</i> equal to DMAMGR_DYNAMIC_ALLOCATE. |
| <i>handle</i>         | DMA or eDMA handle pointer.                                                                                                                               |

Return values

|                                             |                                                                                         |
|---------------------------------------------|-----------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>                      | In dynamic/static channel allocate mechanism, allocate DMAMUX channel successfully.     |
| <i>kStatus_DMAMGR_NoFreeChannel</i>         | In dynamic channel allocate mechanism, all DMAMUX channels has been occupied.           |
| <i>kStatus_DMAMGR_ChannelNotMatchSource</i> | In static channel allocate mechanism, the given channel do not match the given request. |
| <i>kStatus_DMAMGR_ChannelOccupied</i>       | In static channel allocate mechanism, the given channel has been occupied.              |

#### 41.6.4 status\_t DMAMGR\_ReleaseChannel ( *void \* handle* )

This function releases an occupied DMA channel.

## Function Documentation

Parameters

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

Return values

|                                       |                                                                 |
|---------------------------------------|-----------------------------------------------------------------|
| <i>kStatus_Success</i>                | Release the given channel successfully.                         |
| <i>kStatus_DMAMGR_-ChannelNotUsed</i> | The given channel which to be released is not been used before. |

# Chapter 42

## Secured Digital Card/Embedded MultiMedia Card (CARD)

### 42.1 Overview

The Kinetis SDK provides a driver to access the Secured Digital Card and Embedded MultiMedia Card based on the SDHC driver.

### Function groups

This function group implements the SD card functional API.

This function group implements the MMC card functional API.

### Typical use case

```
/* Initialize SDHC. */
sdhcConfig->cardDetectDat3 = false;
sdhcConfig->endianMode = kSDHC_EndianModeLittle;
sdhcConfig->dmaMode = kSDHC_DmaModeAdma2;
sdhcConfig->readWatermarkLevel = 0x80U;
sdhcConfig->writeWatermarkLevel = 0x80U;
SDHC_Init(BOARD_SDHC_BASEADDR, sdhcConfig);

/* Save host information. */
card->host.base = BOARD_SDHC_BASEADDR;
card->host.sourceClock_Hz = CLOCK_GetFreq(BOARD_SDHC_CLKSRC);
card->host.transfer = SDHC_TransferFunction;

/* Init card. */
if (SD_Init(card))
{
    PRINTF("\r\nSD card init failed.\r\n");
}

while (true)
{
    if (kStatus_Success != SD_WriteBlocks(card, g_dataWrite, DATA_BLOCK_START,
        DATA_BLOCK_COUNT))
    {
        PRINTF("Write multiple data blocks failed.\r\n");
    }
    if (kStatus_Success != SD_ReadBlocks(card, g_dataRead, DATA_BLOCK_START, DATA_BLOCK_COUNT))
    {
        PRINTF("Read multiple data blocks failed.\r\n");
    }

    if (kStatus_Success != SD_EraseBlocks(card, DATA_BLOCK_START, DATA_BLOCK_COUNT))
    {
        PRINTF("Erase multiple data blocks failed.\r\n");
    }
}

SD_Deinit(card);

/* Initialize SDHC. */
```

## Overview

```
sdhcConfig->cardDetectDat3 = false;
sdhcConfig->endianMode = kSDHC_EndianModeLittle;
sdhcConfig->dmaMode = kSDHC_DmaModeAdma2;
sdhcConfig->readWatermarkLevel = 0x80U;
sdhcConfig->writeWatermarkLevel = 0x80U;
SDHC_Init(BOARD_SDHC_BASEADDR, sdhcConfig);

/* Save host information.*/
card->host.base = BOARD_SDHC_BASEADDR;
card->host.sourceClock_Hz = CLOCK_GetFreq(BOARD_SDHC_CLKSRC);
card->host.transfer = SDHC_TransferFunction;

/* Init card.*/
if (MMC_Init(card))
{
    PRINTF("\n MMC card init failed \n");
}

while (true)
{
    if (kStatus_Success != MMC_WriteBlocks(card, g_dataWrite, DATA_BLOCK_START,
        DATA_BLOCK_COUNT))
    {
        PRINTF("Write multiple data blocks failed.\r\n");
    }
    if (kStatus_Success != MMC_ReadBlocks(card, g_dataRead, DATA_BLOCK_START,
        DATA_BLOCK_COUNT))
    {
        PRINTF("Read multiple data blocks failed.\r\n");
    }
}
MMC_Deinit(card);
```

## Data Structures

- struct [sd\\_card\\_t](#)  
*SD card state.* [More...](#)
- struct [mmc\\_card\\_t](#)  
*SD card state.* [More...](#)
- struct [mmc\\_boot\\_config\\_t](#)  
*MMC card boot configuration definition.* [More...](#)

## Macros

- #define [FSL\\_SDMMC\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2U, 1U, 1U)) /\*2.1.1\*/
*Driver version.*
- #define [FSL\\_SDMMC\\_DEFAULT\\_BLOCK\\_SIZE](#) (512U)
*Default block size.*

## Enumerations

- enum `_sdmmc_status` {
   
    `kStatus_SDMMC_NotSupportYet` = MAKE\_STATUS(kStatusGroup\_SDMMC, 0U),
   
    `kStatus_SDMMC_TransferFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 1U),
   
    `kStatus_SDMMC_SetCardBlockSizeFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 2U),
   
    `kStatus_SDMMC_HostNotSupport` = MAKE\_STATUS(kStatusGroup\_SDMMC, 3U),
   
    `kStatus_SDMMC_CardNotSupport` = MAKE\_STATUS(kStatusGroup\_SDMMC, 4U),
   
    `kStatus_SDMMC_AllSendCidFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 5U),
   
    `kStatus_SDMMC_SendRelativeAddressFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 6U),
   
    `kStatus_SDMMC_SendCsdFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 7U),
   
    `kStatus_SDMMC_SelectCardFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 8U),
   
    `kStatus_SDMMC_SendScrFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 9U),
   
    `kStatus_SDMMC_SetDataBusWidthFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 10U),
   
    `kStatus_SDMMC_GoIdleFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 11U),
   
    `kStatus_SDMMC_HandShakeOperationConditionFailed`,
   
    `kStatus_SDMMC_SendApplicationCommandFailed`,
   
    `kStatus_SDMMC_SwitchFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 14U),
   
    `kStatus_SDMMC_StopTransmissionFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 15U),
   
    `kStatus_SDMMC_WaitWriteCompleteFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 16U),
   
    `kStatus_SDMMC_SetBlockCountFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 17U),
   
    `kStatus_SDMMC_SetRelativeAddressFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 18U),
   
    `kStatus_SDMMC_SwitchHighSpeedFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 19U),
   
    `kStatus_SDMMC_SendExtendedCsdFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 20U),
   
    `kStatus_SDMMC_ConfigureBootFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 21U),
   
    `kStatus_SDMMC_ConfigureExtendedCsdFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 22U),
   
    `kStatus_SDMMC_EnableHighCapacityEraseFailed`,
   
    `kStatus_SDMMC_SendTestPatternFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 24U),
   
    `kStatus_SDMMC_ReceiveTestPatternFailed` = MAKE\_STATUS(kStatusGroup\_SDMMC, 25U) }
   
    *SD/MMC card API's running status.*
- enum `_sd_card_flag` {
   
    `kSD_SupportHighCapacityFlag` = (1U << 1U),
   
    `kSD_Support4BitWidthFlag` = (1U << 2U),
   
    `kSD_SupportSdhcFlag` = (1U << 3U),
   
    `kSD_SupportSdxcFlag` = (1U << 4U) }
   
    *SD card flags.*
- enum `_mmc_card_flag` {
   
    `kMMC_SupportHighCapacityFlag` = (1U << 0U),
   
    `kMMC_SupportHighSpeedFlag` = (1U << 1U),
   
    `kMMC_SupportHighSpeed52MHZFlag` = (1U << 2U),
   
    `kMMC_SupportHighSpeed26MHZFlag` = (1U << 3U),
   
    `kMMC_SupportAlternateBootFlag` = (1U << 4U) }
   
    *MMC card flags.*

## Data Structure Documentation

### SDCARD Function

- status\_t [SD\\_Init](#) ([sd\\_card\\_t](#) \*card)  
*Initialize the card on a specific host controller.*
- void [SD\\_Deinit](#) ([sd\\_card\\_t](#) \*card)  
*Deinitialize the card.*
- bool [SD\\_CheckReadOnly](#) ([sd\\_card\\_t](#) \*card)  
*Check whether the card is write-protected.*
- status\_t [SD\\_ReadBlocks](#) ([sd\\_card\\_t](#) \*card, [uint8\\_t](#) \*buffer, [uint32\\_t](#) startBlock, [uint32\\_t](#) blockCount)  
*Read blocks from the specific card.*
- status\_t [SD\\_WriteBlocks](#) ([sd\\_card\\_t](#) \*card, const [uint8\\_t](#) \*buffer, [uint32\\_t](#) startBlock, [uint32\\_t](#) blockCount)  
*Write blocks of data to the specific card.*
- status\_t [SD\\_EraseBlocks](#) ([sd\\_card\\_t](#) \*card, [uint32\\_t](#) startBlock, [uint32\\_t](#) blockCount)  
*Erase blocks of the specific card.*

### MMCCARD Function

- status\_t [MMC\\_Init](#) ([mmc\\_card\\_t](#) \*card)  
*Initialize the MMC card.*
- void [MMC\\_Deinit](#) ([mmc\\_card\\_t](#) \*card)  
*Deinitialize the card.*
- bool [MMC\\_CheckReadOnly](#) ([mmc\\_card\\_t](#) \*card)  
*Check if the card is read only.*
- status\_t [MMC\\_ReadBlocks](#) ([mmc\\_card\\_t](#) \*card, [uint8\\_t](#) \*buffer, [uint32\\_t](#) startBlock, [uint32\\_t](#) blockCount)  
*Read data blocks from the card.*
- status\_t [MMC\\_WriteBlocks](#) ([mmc\\_card\\_t](#) \*card, const [uint8\\_t](#) \*buffer, [uint32\\_t](#) startBlock, [uint32\\_t](#) blockCount)  
*Write data blocks to the card.*
- status\_t [MMC\\_EraseGroups](#) ([mmc\\_card\\_t](#) \*card, [uint32\\_t](#) startGroup, [uint32\\_t](#) endGroup)  
*Erase groups of the card.*
- status\_t [MMC\\_SelectPartition](#) ([mmc\\_card\\_t](#) \*card, [mmc\\_access\\_partition\\_t](#) partitionNumber)  
*Select the partition to access.*
- status\_t [MMC\\_SetBootConfig](#) ([mmc\\_card\\_t](#) \*card, const [mmc\\_boot\\_config\\_t](#) \*config)  
*Configure boot activity of the card.*

## 42.2 Data Structure Documentation

### 42.2.1 struct [sd\\_card\\_t](#)

Define the card structure including the necessary fields to identify and describe the card.

#### Data Fields

- [sdhc\\_host\\_t](#) [host](#)  
*Host information.*
- [uint32\\_t](#) [busClock\\_Hz](#)

- **SD bus clock frequency united in Hz.**
- **uint32\_t relativeAddress**  
*Relative address of the card.*
- **uint32\_t version**  
*Card version.*
- **uint32\_t flags**  
*Flags in \_sd\_card\_flag.*
- **uint32\_t rawCid [4U]**  
*Raw CID content.*
- **uint32\_t rawCsd [4U]**  
*Raw CSD content.*
- **uint32\_t rawScr [2U]**  
*Raw SCR content.*
- **uint32\_t ocr**  
*Raw OCR content.*
- **sd\_cid\_t cid**  
*CID.*
- **sd\_csd\_t csd**  
*CSD.*
- **sd\_scr\_t scr**  
*SCR.*
- **uint32\_t blockCount**  
*Card total block number.*
- **uint32\_t blockSize**  
*Card block size.*

#### 42.2.2 struct mmc\_card\_t

Define the card structure including the necessary fields to identify and describe the card.

#### Data Fields

- **sdhc\_host\_t host**  
*Host information.*
- **uint32\_t busClock\_Hz**  
*MMC bus clock united in Hz.*
- **uint32\_t relativeAddress**  
*Relative address of the card.*
- **bool enablePreDefinedBlockCount**  
*Enable PRE-DEFINED block count when read/write.*
- **uint32\_t flags**  
*Capability flag in \_mmc\_card\_flag.*
- **uint32\_t rawCid [4U]**  
*Raw CID content.*
- **uint32\_t rawCsd [4U]**  
*Raw CSD content.*
- **uint32\_t rawExtendedCsd [MMC\_EXTENDED\_CSD\_BYTES/4U]**  
*Raw MMC Extended CSD content.*

## Enumeration Type Documentation

- `uint32_t ocr`  
*Raw OCR content.*
- `mmc_cid_t cid`  
*CID.*
- `mmc_csd_t csd`  
*CSD.*
- `mmc_extended_csd_t extendedCsd`  
*Extended CSD.*
- `uint32_t blockSize`  
*Card block size.*
- `uint32_t userPartitionBlocks`  
*Card total block number in user partition.*
- `uint32_t bootPartitionBlocks`  
*Boot partition size united as block size.*
- `uint32_t eraseGroupBlocks`  
*Erase group size united as block size.*
- `mmc_access_partition_t currentPartition`  
*Current access partition.*
- `mmc_voltage_window_t hostVoltageWindow`  
*Host voltage window.*

### 42.2.3 `struct mmc_boot_config_t`

#### Data Fields

- `bool enableBootAck`  
*Enable boot ACK.*
- `mmc_boot_partition_enable_t bootPartition`  
*Boot partition.*
- `bool retainBootBusWidth`  
*If retain boot bus width.*
- `mmc_data_bus_width_t bootDataBusWidth`  
*Boot data bus width.*

## 42.3 Macro Definition Documentation

### 42.3.1 `#define FSL_SDMMC_DRIVER_VERSION (MAKE_VERSION(2U, 1U, 1U))` /\*2.1.1\*/

## 42.4 Enumeration Type Documentation

### 42.4.1 `enum _sdmmc_status`

Enumerator

- `kStatus_SDMMC_NotSupportYet` Haven't supported.  
`kStatus_SDMMC_TransferFailed` Send command failed.  
`kStatus_SDMMC_SetCardBlockSizeFailed` Set block size failed.

*kStatus\_SDMMC\_HostNotSupport* Host doesn't support.  
*kStatus\_SDMMC\_CardNotSupport* Card doesn't support.  
*kStatus\_SDMMC\_AllSendCidFailed* Send CID failed.  
*kStatus\_SDMMC\_SendRelativeAddressFailed* Send relative address failed.  
*kStatus\_SDMMC\_SendCsdFailed* Send CSD failed.  
*kStatus\_SDMMC\_SelectCardFailed* Select card failed.  
*kStatus\_SDMMC\_SendScrFailed* Send SCR failed.  
*kStatus\_SDMMC\_SetDataBusWidthFailed* Set bus width failed.  
*kStatus\_SDMMC\_GoIdleFailed* Go idle failed.  
*kStatus\_SDMMC\_HandShakeOperationConditionFailed* Send Operation Condition failed.  
*kStatus\_SDMMC\_SendApplicationCommandFailed* Send application command failed.  
*kStatus\_SDMMC\_SwitchFailed* Switch command failed.  
*kStatus\_SDMMC\_StopTransmissionFailed* Stop transmission failed.  
*kStatus\_SDMMC\_WaitWriteCompleteFailed* Wait write complete failed.  
*kStatus\_SDMMC\_SetBlockCountFailed* Set block count failed.  
*kStatus\_SDMMC\_SetRelativeAddressFailed* Set relative address failed.  
*kStatus\_SDMMC\_SwitchHighSpeedFailed* Switch high speed failed.  
*kStatus\_SDMMC\_SendExtendedCsdFailed* Send EXT\_CSD failed.  
*kStatus\_SDMMC\_ConfigureBootFailed* Configure boot failed.  
*kStatus\_SDMMC\_ConfigureExtendedCsdFailed* Configure EXT\_CSD failed.  
*kStatus\_SDMMC\_EnableHighCapacityEraseFailed* Enable high capacity erase failed.  
*kStatus\_SDMMC\_SendTestPatternFailed* Send test pattern failed.  
*kStatus\_SDMMC\_ReceiveTestPatternFailed* Receive test pattern failed.

#### 42.4.2 enum\_sd\_card\_flag

Enumerator

*kSD\_SupportHighCapacityFlag* Support high capacity.  
*kSD\_Support4BitWidthFlag* Support 4-bit data width.  
*kSD\_SupportSdhcFlag* Card is SDHC.  
*kSD\_SupportSdxcFlag* Card is SDXC.

#### 42.4.3 enum\_mmc\_card\_flag

Enumerator

*kMMC\_SupportHighCapacityFlag* Support high capacity.  
*kMMC\_SupportHighSpeedFlag* Support high speed.  
*kMMC\_SupportHighSpeed52MHZFlag* Support high speed 52MHZ.  
*kMMC\_SupportHighSpeed26MHZFlag* Support high speed 26MHZ.  
*kMMC\_SupportAlternateBootFlag* Support alternate boot.

## Function Documentation

### 42.5 Function Documentation

#### 42.5.1 **status\_t SD\_Init( sd\_card\_t \* card )**

This function initializes the card on a specific host controller.

## Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

## Return values

|                                                     |                                  |
|-----------------------------------------------------|----------------------------------|
| <i>kStatus_SDMMC_Go-IdleFailed</i>                  | Go idle failed.                  |
| <i>kStatus_SDMMC_Not-SupportYet</i>                 | Card not support.                |
| <i>kStatus_SDMMC_Send-OperationCondition-Failed</i> | Send operation condition failed. |
| <i>kStatus_SDMMC_All-SendCidFailed</i>              | Send CID failed.                 |
| <i>kStatus_SDMMC_Send-RelativeAddressFailed</i>     | Send relative address failed.    |
| <i>kStatus_SDMMC_Send-CsdFailed</i>                 | Send CSD failed.                 |
| <i>kStatus_SDMMC_Select-CardFailed</i>              | Send SELECT_CARD command failed. |
| <i>kStatus_SDMMC_Send-ScrFailed</i>                 | Send SCR failed.                 |
| <i>kStatus_SDMMC_SetBus-WidthFailed</i>             | Set bus width failed.            |
| <i>kStatus_SDMMC_Switch-HighSpeedFailed</i>         | Switch high speed failed.        |
| <i>kStatus_SDMMC_Set-CardBlockSizeFailed</i>        | Set card block size failed.      |
| <i>kStatus_Success</i>                              | Operate successfully.            |

**42.5.2 void SD\_Deinit ( sd\_card\_t \* *card* )**

This function deinitializes the specific card.

## Function Documentation

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

### 42.5.3 bool SD\_CheckReadOnly ( *sd\_card\_t \* card* )

This function checks if the card is write-protected via CSD register.

Parameters

|             |                    |
|-------------|--------------------|
| <i>card</i> | The specific card. |
|-------------|--------------------|

Return values

|              |                       |
|--------------|-----------------------|
| <i>true</i>  | Card is read only.    |
| <i>false</i> | Card isn't read only. |

### 42.5.4 status\_t SD\_ReadBlocks ( *sd\_card\_t \* card, uint8\_t \* buffer, uint32\_t startBlock, uint32\_t blockCount* )

This function reads blocks from specific card, with default block size defined by SDHC\_CARD\_DEFAULT\_BLOCK\_SIZE.

Parameters

|                   |                                             |
|-------------------|---------------------------------------------|
| <i>card</i>       | Card descriptor.                            |
| <i>buffer</i>     | The buffer to save the data read from card. |
| <i>startBlock</i> | The start block index.                      |
| <i>blockCount</i> | The number of blocks to read.               |

Return values

|                                     |                   |
|-------------------------------------|-------------------|
| <i>kStatus_InvalidArgument</i>      | Invalid argument. |
| <i>kStatus_SDMMC_CardNotSupport</i> | Card not support. |

|                                              |                           |
|----------------------------------------------|---------------------------|
| <i>kStatus_SDMMC_NotSupportYet</i>           | Not support now.          |
| <i>kStatus_SDMMC_WaitWriteCompleteFailed</i> | Send status failed.       |
| <i>kStatus_SDMMC_TransferFailed</i>          | Transfer failed.          |
| <i>kStatus_SDMMC_StopTransmissionFailed</i>  | Stop transmission failed. |
| <i>kStatus_Success</i>                       | Operate successfully.     |

#### 42.5.5 **status\_t SD\_WriteBlocks ( *sd\_card\_t \* card, const uint8\_t \* buffer, uint32\_t startBlock, uint32\_t blockCount* )**

This function writes blocks to specific card, with default block size 512 bytes.

Parameters

|                   |                                                        |
|-------------------|--------------------------------------------------------|
| <i>card</i>       | Card descriptor.                                       |
| <i>buffer</i>     | The buffer holding the data to be written to the card. |
| <i>startBlock</i> | The start block index.                                 |
| <i>blockCount</i> | The number of blocks to write.                         |

Return values

|                                              |                     |
|----------------------------------------------|---------------------|
| <i>kStatus_InvalidArgument</i>               | Invalid argument.   |
| <i>kStatus_SDMMC_NotSupportYet</i>           | Not support now.    |
| <i>kStatus_SDMMC_CardNotSupport</i>          | Card not support.   |
| <i>kStatus_SDMMC_WaitWriteCompleteFailed</i> | Send status failed. |
| <i>kStatus_SDMMC_TransferFailed</i>          | Transfer failed.    |

## Function Documentation

|                                             |                           |
|---------------------------------------------|---------------------------|
| <i>kStatus_SDMMC_StopTransmissionFailed</i> | Stop transmission failed. |
| <i>kStatus_Success</i>                      | Operate successfully.     |

### 42.5.6 **status\_t SD\_EraseBlocks ( *sd\_card\_t* \* *card*, *uint32\_t* *startBlock*, *uint32\_t* *blockCount* )**

This function erases blocks of a specific card, with default block size 512 bytes.

#### Parameters

|                   |                                |
|-------------------|--------------------------------|
| <i>card</i>       | Card descriptor.               |
| <i>startBlock</i> | The start block index.         |
| <i>blockCount</i> | The number of blocks to erase. |

#### Return values

|                                              |                       |
|----------------------------------------------|-----------------------|
| <i>kStatus_InvalidArgument</i>               | Invalid argument.     |
| <i>kStatus_SDMMC_WaitWriteCompleteFailed</i> | Send status failed.   |
| <i>kStatus_SDMMC_TransferFailed</i>          | Transfer failed.      |
| <i>kStatus_SDMMC_WaitWriteCompleteFailed</i> | Send status failed.   |
| <i>kStatus_Success</i>                       | Operate successfully. |

### 42.5.7 **status\_t MMC\_Init ( *mmc\_card\_t* \* *card* )**

#### Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

#### Return values

|                                                   |                                  |
|---------------------------------------------------|----------------------------------|
| <i>kStatus_SDMMC_GoIdleFailed</i>                 | Go idle failed.                  |
| <i>kStatus_SDMMC_SendOperationConditionFailed</i> | Send operation condition failed. |
| <i>kStatus_SDMMC_AllSendCidFailed</i>             | Send CID failed.                 |
| <i>kStatus_SDMMC_SetRelativeAddressFailed</i>     | Set relative address failed.     |
| <i>kStatus_SDMMC_SendCsdFailed</i>                | Send CSD failed.                 |
| <i>kStatus_SDMMC_CardNotSupport</i>               | Card not support.                |
| <i>kStatus_SDMMC_SelectCardFailed</i>             | Send SELECT_CARD command failed. |
| <i>kStatus_SDMMC_SendExtendedCsdFailed</i>        | Send EXT_CSD failed.             |
| <i>kStatus_SDMMC_SetBusWidthFailed</i>            | Set bus width failed.            |
| <i>kStatus_SDMMC_SwitchHighSpeedFailed</i>        | Switch high speed failed.        |
| <i>kStatus_SDMMC_SetCardBlockSizeFailed</i>       | Set card block size failed.      |
| <i>kStatus_Success</i>                            | Operate successfully.            |

#### 42.5.8 void MMC\_Deinit ( mmc\_card\_t \* *card* )

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

#### 42.5.9 bool MMC\_CheckReadOnly ( mmc\_card\_t \* *card* )

## Function Documentation

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

Return values

|              |                       |
|--------------|-----------------------|
| <i>true</i>  | Card is read only.    |
| <i>false</i> | Card isn't read only. |

### 42.5.10 **status\_t MMC\_ReadBlocks ( mmc\_card\_t \* *card*, uint8\_t \* *buffer*, uint32\_t *startBlock*, uint32\_t *blockCount* )**

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>card</i>       | Card descriptor.              |
| <i>buffer</i>     | The buffer to save data.      |
| <i>startBlock</i> | The start block index.        |
| <i>blockCount</i> | The number of blocks to read. |

Return values

|                                              |                           |
|----------------------------------------------|---------------------------|
| <i>kStatus_InvalidArgument</i>               | Invalid argument.         |
| <i>kStatus_SDMMC_Card-NotSupport</i>         | Card not support.         |
| <i>kStatus_SDMMC_Set-BlockCountFailed</i>    | Set block count failed.   |
| <i>kStatus_SDMMC_TransferFailed</i>          | Transfer failed.          |
| <i>kStatus_SDMMC_Stop-TransmissionFailed</i> | Stop transmission failed. |
| <i>kStatus_Success</i>                       | Operate successfully.     |

### 42.5.11 **status\_t MMC\_WriteBlocks ( mmc\_card\_t \* *card*, const uint8\_t \* *buffer*, uint32\_t *startBlock*, uint32\_t *blockCount* )**

Parameters

|                   |                                 |
|-------------------|---------------------------------|
| <i>card</i>       | Card descriptor.                |
| <i>buffer</i>     | The buffer to save data blocks. |
| <i>startBlock</i> | Start block number to write.    |
| <i>blockCount</i> | Block count.                    |

Return values

|                                              |                           |
|----------------------------------------------|---------------------------|
| <i>kStatus_InvalidArgument</i>               | Invalid argument.         |
| <i>kStatus_SDMMC_NotSupportYet</i>           | Not support now.          |
| <i>kStatus_SDMMC_SetBlockCountFailed</i>     | Set block count failed.   |
| <i>kStatus_SDMMC_WaitWriteCompleteFailed</i> | Send status failed.       |
| <i>kStatus_SDMMC_TransferFailed</i>          | Transfer failed.          |
| <i>kStatus_SDMMC_StopTransmissionFailed</i>  | Stop transmission failed. |
| <i>kStatus_Success</i>                       | Operate successfully.     |

#### 42.5.12 **status\_t MMC\_EraseGroups ( mmc\_card\_t \* *card*, uint32\_t *startGroup*, uint32\_t *endGroup* )**

Erase group is the smallest erase unit in MMC card. The erase range is [startGroup, endGroup].

Parameters

|                   |                     |
|-------------------|---------------------|
| <i>card</i>       | Card descriptor.    |
| <i>startGroup</i> | Start group number. |
| <i>endGroup</i>   | End group number.   |

Return values

## Function Documentation

|                                               |                       |
|-----------------------------------------------|-----------------------|
| <i>kStatus_InvalidArgument</i>                | Invalid argument.     |
| <i>kStatus_SDMMC_Wait-WriteCompleteFailed</i> | Send status failed.   |
| <i>kStatus_SDMMC_TransferFailed</i>           | Transfer failed.      |
| <i>kStatus_Success</i>                        | Operate successfully. |

### 42.5.13 **status\_t MMC\_SelectPartition ( mmc\_card\_t \* *card*, mmc\_access\_partition\_t *partitionNumber* )**

Parameters

|                         |                       |
|-------------------------|-----------------------|
| <i>card</i>             | Card descriptor.      |
| <i>partition-Number</i> | The partition number. |

Return values

|                                                  |                           |
|--------------------------------------------------|---------------------------|
| <i>kStatus_SDMMC_ConfigureExtendedCsd-Failed</i> | Configure EXT_CSD failed. |
| <i>kStatus_Success</i>                           | Operate successfully.     |

### 42.5.14 **status\_t MMC\_SetBootConfig ( mmc\_card\_t \* *card*, const mmc\_boot\_config\_t \* *config* )**

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>card</i>   | Card descriptor.              |
| <i>config</i> | Boot configuration structure. |

Return values

|                                                   |                           |
|---------------------------------------------------|---------------------------|
| <i>kStatus_SDMMC_Not-SupportYet</i>               | Not support now.          |
| <i>kStatus_SDMMC_-ConfigureExtendedCsd-Failed</i> | Configure EXT_CSD failed. |
| <i>kStatus_SDMMC_-ConfigureBootFailed</i>         | Configure boot failed.    |
| <i>kStatus_Success</i>                            | Operate successfully.     |

## Function Documentation

# Chapter 43

## SPI based Secured Digital Card (SDSPI)

### 43.1 Overview

The KSDK provides a driver to access the Secured Digital Card based on the SPI driver.

### Function groups

This function group implements the SD card functional API in the SPI mode.

### Typical use case

```
/* SPI_Init(). */

/* Register the SDSPI driver callback. */

/* Initializes card. */
if (kStatus_Success != SDSPI_Init(card))
{
    SDSPI_Deinit(card)
    return;
}

/* Read/Write card */
memset(g_testWriteBuffer, 0x17U, sizeof(g_testWriteBuffer));

while (true)
{
    memset(g_testReadBuffer, 0U, sizeof(g_testReadBuffer));

    SDSPI_WriteBlocks(card, g_testWriteBuffer, TEST_START_BLOCK, TEST_BLOCK_COUNT);

    SDSPI_ReadBlocks(card, g_testReadBuffer, TEST_START_BLOCK, TEST_BLOCK_COUNT);

    if (memcmp(g_testReadBuffer, g_testReadBuffer, sizeof(g_testWriteBuffer)))
    {
        break;
    }
}
```

### Data Structures

- struct `sdspi_command_t`  
*SDSPI command.* [More...](#)
- struct `sdspi_host_t`  
*SDSPI host state.* [More...](#)
- struct `sdspi_card_t`  
*SD Card Structure.* [More...](#)

## Overview

### Enumerations

- enum `_sdspi_status` {  
    `kStatus_SDSPI_SetFrequencyFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 0U),  
    `kStatus_SDSPI_ExchangeFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 1U),  
    `kStatus_SDSPI_WaitReadyFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 2U),  
    `kStatus_SDSPI_ResponseError` = MAKE\_STATUS(kStatusGroup\_SDSPI, 3U),  
    `kStatus_SDSPI_WriteProtected` = MAKE\_STATUS(kStatusGroup\_SDSPI, 4U),  
    `kStatus_SDSPI_GoIdleFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 5U),  
    `kStatus_SDSPI_SendCommandFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 6U),  
    `kStatus_SDSPI_ReadFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 7U),  
    `kStatus_SDSPI_WriteFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 8U),  
    `kStatus_SDSPI_SendInterfaceConditionFailed`,  
    `kStatus_SDSPI_SendOperationConditionFailed`,  
    `kStatus_SDSPI_ReadOcrFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 11U),  
    `kStatus_SDSPI_SetBlockSizeFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 12U),  
    `kStatus_SDSPI_SendCsdFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 13U),  
    `kStatus_SDSPI_SendCidFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 14U),  
    `kStatus_SDSPI_StopTransmissionFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 15U),  
    `kStatus_SDSPI_SendApplicationCommandFailed` }

*SDSPI API status.*

- enum `_sdspi_card_flag` {  
    `kSDSPI_SupportHighCapacityFlag` = (1U << 0U),  
    `kSDSPI_SupportSdhcFlag` = (1U << 1U),  
    `kSDSPI_SupportSdxcFlag` = (1U << 2U),  
    `kSDSPI_SupportSdscFlag` = (1U << 3U) }

*SDSPI card flag.*

- enum `sdspi_response_type_t` {  
    `kSDSPI_ResponseTypeR1` = 0U,  
    `kSDSPI_ResponseTypeR1b` = 1U,  
    `kSDSPI_ResponseTypeR2` = 2U,  
    `kSDSPI_ResponseTypeR3` = 3U,  
    `kSDSPI_ResponseTypeR7` = 4U }

*SDSPI response type.*

### SDSPI Function

- `status_t SDSPI_Init (sdspi_card_t *card)`  
*Initialize the card on a specific SPI instance.*
- `void SDSPI_Deinit (sdspi_card_t *card)`  
*Deinitialize the card.*
- `bool SDSPI_CheckReadOnly (sdspi_card_t *card)`  
*Check whether the card is write-protected.*
- `status_t SDSPI_ReadBlocks (sdspi_card_t *card, uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)`  
*Read blocks from the specific card.*
- `status_t SDSPI_WriteBlocks (sdspi_card_t *card, uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)`

*Write blocks of data to the specific card.*

## 43.2 Data Structure Documentation

### 43.2.1 struct sdspi\_command\_t

#### Data Fields

- `uint8_t index`  
*Command index.*
- `uint32_t argument`  
*Command argument.*
- `uint8_t responseType`  
*Response type.*
- `uint8_t response [5U]`  
*Response content.*

### 43.2.2 struct sdspi\_host\_t

#### Data Fields

- `uint32_t busBaudRate`  
*Bus baud rate.*
- `status_t(* setFrequency )(uint32_t frequency)`  
*Set frequency of SPI.*
- `status_t(* exchange )(uint8_t *in, uint8_t *out, uint32_t size)`  
*Exchange data over SPI.*
- `uint32_t(* getCurrentMilliseconds )(void)`  
*Get current time in milliseconds.*

### 43.2.3 struct sdspi\_card\_t

Define the card structure including the necessary fields to identify and describe the card.

#### Data Fields

- `sdspi_host_t * host`  
*Host state information.*
- `uint32_t relativeAddress`  
*Relative address of the card.*
- `uint32_t flags`  
*Flags defined in \_sdspi\_card\_flag.*
- `uint8_t rawCid [16U]`  
*Raw CID content.*
- `uint8_t rawCsd [16U]`

## Enumeration Type Documentation

- `uint8_t rawScr [8U]`  
*Raw CSD content.*
- `uint32_t ocr`  
*Raw OCR content.*
- `sd_cid_t cid`  
*CID.*
- `sd_csd_t csd`  
*CSD.*
- `sd_scr_t scr`  
*SCR.*
- `uint32_t blockCount`  
*Card total block number.*
- `uint32_t blockSize`  
*Card block size.*

### 43.2.3.0.0.26 Field Documentation

#### 43.2.3.0.0.26.1 `uint32_t sdspi_card_t::flags`

## 43.3 Enumeration Type Documentation

### 43.3.1 `enum _sdspi_status`

Enumerator

- `kStatus_SDSPI_SetFrequencyFailed` Set frequency failed.
- `kStatus_SDSPI_ExchangeFailed` Exchange data on SPI bus failed.
- `kStatus_SDSPI_WaitReadyFailed` Wait card ready failed.
- `kStatus_SDSPI_ResponseError` Response is error.
- `kStatus_SDSPI_WriteProtected` Write protected.
- `kStatus_SDSPI_GoIdleFailed` Go idle failed.
- `kStatus_SDSPI_SendCommandFailed` Send command failed.
- `kStatus_SDSPI_ReadFailed` Read data failed.
- `kStatus_SDSPI_WriteFailed` Write data failed.
- `kStatus_SDSPI_SendInterfaceConditionFailed` Send interface condition failed.
- `kStatus_SDSPI_SendOperationConditionFailed` Send operation condition failed.
- `kStatus_SDSPI_ReadOcrFailed` Read OCR failed.
- `kStatus_SDSPI_SetBlockSizeFailed` Set block size failed.
- `kStatus_SDSPI_SendCsdFailed` Send CSD failed.
- `kStatus_SDSPI_SendCidFailed` Send CID failed.
- `kStatus_SDSPI_StopTransmissionFailed` Stop transmission failed.
- `kStatus_SDSPI_SendApplicationCommandFailed` Send application command failed.

### 43.3.2 enum \_sdspi\_card\_flag

Enumerator

*kSDSPI\_SupportHighCapacityFlag* Card is high capacity.

*kSDSPI\_SupportSdhcFlag* Card is SDHC.

*kSDSPI\_SupportSdxcFlag* Card is SDXC.

*kSDSPI\_SupportSdscFlag* Card is SDSC.

### 43.3.3 enum sdspi\_response\_type\_t

Enumerator

*kSDSPI\_ResponseOfTypeR1* Response 1.

*kSDSPI\_ResponseOfTypeR1b* Response 1 with busy.

*kSDSPI\_ResponseOfTypeR2* Response 2.

*kSDSPI\_ResponseOfTypeR3* Response 3.

*kSDSPI\_ResponseOfTypeR7* Response 7.

## 43.4 Function Documentation

### 43.4.1 status\_t SDSPI\_Init ( *sdspi\_card\_t \* card* )

This function initializes the card on a specific SPI instance.

Parameters

|             |                 |
|-------------|-----------------|
| <i>card</i> | Card descriptor |
|-------------|-----------------|

Return values

|                                                    |                                  |
|----------------------------------------------------|----------------------------------|
| <i>kStatus_SD SPI_SetFrequencyFailed</i>           | Set frequency failed.            |
| <i>kStatus_SD SPI_GoIdleFailed</i>                 | Go idle failed.                  |
| <i>kStatus_SD SPI_SendInterfaceConditionFailed</i> | Send interface condition failed. |

## Function Documentation

|                                                   |                                  |
|---------------------------------------------------|----------------------------------|
| <i>kStatus_SDSPI_SendOperationConditionFailed</i> | Send operation condition failed. |
| <i>kStatus_Timeout</i>                            | Send command timeout.            |
| <i>kStatus_SDSPI_NotSupportYet</i>                | Not support yet.                 |
| <i>kStatus_SDSPI_ReadOcrFailed</i>                | Read OCR failed.                 |
| <i>kStatus_SDSPI_SetBlockSizeFailed</i>           | Set block size failed.           |
| <i>kStatus_SDSPI_SendCsdFailed</i>                | Send CSD failed.                 |
| <i>kStatus_SDSPI_SendCidFailed</i>                | Send CID failed.                 |
| <i>kStatus_Success</i>                            | Operate successfully.            |

### 43.4.2 void SDSPI\_Deinit ( *sdspi\_card\_t \* card* )

This function deinitializes the specific card.

Parameters

|             |                 |
|-------------|-----------------|
| <i>card</i> | Card descriptor |
|-------------|-----------------|

### 43.4.3 bool SDSPI\_CheckReadOnly ( *sdspi\_card\_t \* card* )

This function checks if the card is write-protected via CSD register.

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

Return values

---

|              |                       |
|--------------|-----------------------|
| <i>true</i>  | Card is read only.    |
| <i>false</i> | Card isn't read only. |

#### 43.4.4 **status\_t SDSPI\_ReadBlocks ( *sdspi\_card\_t \* card, uint8\_t \* buffer, uint32\_t startBlock, uint32\_t blockCount* )**

This function reads blocks from specific card.

Parameters

|                   |                                            |
|-------------------|--------------------------------------------|
| <i>card</i>       | Card descriptor.                           |
| <i>buffer</i>     | the buffer to hold the data read from card |
| <i>startBlock</i> | the start block index                      |
| <i>blockCount</i> | the number of blocks to read               |

Return values

|                                              |                           |
|----------------------------------------------|---------------------------|
| <i>kStatus_SDSPI_Send-CommandFailed</i>      | Send command failed.      |
| <i>kStatus_SDSPI_Read-Failed</i>             | Read data failed.         |
| <i>kStatus_SDSPI_Stop-TransmissionFailed</i> | Stop transmission failed. |
| <i>kStatus_Success</i>                       | Operate successfully.     |

#### 43.4.5 **status\_t SDSPI\_WriteBlocks ( *sdspi\_card\_t \* card, uint8\_t \* buffer, uint32\_t startBlock, uint32\_t blockCount* )**

This function writes blocks to specific card

Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>card</i>   | Card descriptor.                                      |
| <i>buffer</i> | the buffer holding the data to be written to the card |

## Function Documentation

|                   |                               |
|-------------------|-------------------------------|
| <i>startBlock</i> | the start block index         |
| <i>blockCount</i> | the number of blocks to write |

Return values

|                                        |                                      |
|----------------------------------------|--------------------------------------|
| <i>kStatus_SDSPI_WriteProtected</i>    | Card is write protected.             |
| <i>kStatus_SDSPI_SendCommandFailed</i> | Send command failed.                 |
| <i>kStatus_SDSPI_ResponseError</i>     | Response is error.                   |
| <i>kStatus_SDSPI_WriteFailed</i>       | Write data failed.                   |
| <i>kStatus_SDSPI_ExchangeFailed</i>    | Exchange data over SPI failed.       |
| <i>kStatus_SDSPI_WaitReadyFailed</i>   | Wait card to be ready status failed. |
| <i>kStatus_Success</i>                 | Operate successfully.                |

**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 Freescale 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.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale 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 Freescale 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. Freescale does not convey any license under its patent rights nor the rights of others. Freescale 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).

Freescale, the Freescale logo, Kinetis, Processor Expert are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Tower is a trademark of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. ARM, ARM Powered logo, µVision, Keil and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2016 Freescale Semiconductor, Inc.

Document Number: KSDKKL8220APIRM

Rev. 0

Jun 2016

